Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Tietojärjestelmien kehitys

Sami Anttonen

Tekoälyn suunnittelu ja toteutus mobiilipeliin

Opinnäytetyö 2013

Tiivistelmä

Sami Anttonen
Tekoälyn suunnittelu ja toteutus mobiilipeliin, 69 sivua, 2 liitettä
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Tietojärjestelmien kehitys
Opinnäytetyö 2013
Ohjaajat: lehtori Mikko Huhtanen, Saimaan ammattikorkeakoulu
toimitusjohtaja Sami Repo, Jontka osk
peliohjelmoija Mikko Laitinen, Jontka osk

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa tekoäly Windows Phone -mobiilikäyttöjärjestelmälle suunnattuun Corrupted Cultura -peliin. Corrupted Cultura on 2D:nä toteutettava toiminta-genren mobiilipeli, ja siinä tekoäly esiintyy lähinnä vihollishahmojen liikkumisena ja päätöksentekona. Opinnäytetyön asiakkaana oli lappeenrantalainen pelialan yritys Jontka osk.

Työ eteni sovelluskehitysprojektina ketterän ohjelmistokehityksen keinoja hyödyntäen. Ohjelmointi tapahtui olio-ohjelmoinnin periaatteiden mukaan käyttäen C#-ohjelmointikieltä ja XNA-ohjelmistokehystä. Opinnäytetyön teoriaosan sekä pohjan suunnittelulle ja ohjelmointityölle muodostivat erilaiset tekoälyä ja Windows Phone -sovelluskehitystä käsittelevät kirjalliset teokset ja oppaat sekä internetlähteet. Suurin osa raportista on tekoälyn toteutuksen ja siihen liittyvien ratkaisujen raportointia.

Opinnäytetyön tuloksena asiakkaalle tuotettiin toimiva ja vaatimusten mukainen tekoäly, jolla voidaan luoda eri tavoin käyttäytyviä hahmoja. Tekoäly mahdollistaa Corrupted Culturan vihollishahmojen liikkumisen ja yksinkertaisen päätöksenteon. Opinnäytetyössä käsiteltiin pelitekoälyä, XNA-ohjelmistokehystä sekä peliohjelmoinnin näkökulmasta Windows Phone -sovelluskehitystä. Tämä opinnäytetyö liitteineen toimii asiakkaalle tekoälyyn liittyvien teknisten ratkaisujen dokumentaationa.

Asiasanat: Tekoäly, XNA, C#, peliohjelmointi, mobiilisovelluskehitys, Windows Phone, Microsoft

Abstract

Sami Anttonen
Design and implementation of artificial intelligence for a mobile game,
69 pages, 2 appendices
Saimaa University of Applied Sciences
Technology Lappeenranta
Degree Programme in Information Technology
Information System Development
Bachelor´s Thesis 2013
Instructors: Senior Lecturer Mikko Huhtanen, Saimaa University of Applied Sciences
CEO Sami Repo, Jontka osk
Senior Game Developer Mikko Laitinen, Jontka osk

The purpose of this thesis was to design and implement artificial intelligence for the game called Corrupted Cultura. Corrupted Cultura is a 2D action-genre mobile game and it is designed for Windows Phone -mobile devices. The artificial intelligence in the game contains decision making and movement of enemy characters. The customer of this thesis was the game company Jontka cooperative from Lappeenranta.

This study was realized as a software development project using values and methods of Agile software development. Programming was done using C# programming language and XNA framework with principles of object-oriented programming. Theoretical knowledge for this thesis and development process were collected from literal works, guide books and the Internet.

The final result of this thesis was a working artificial intelligence that meets the customer's requirements. With the implemented artificial intelligence the enemy characters of Corrupted Cultura are able to decision making and movement. The artificial intelligence also gives different kinds of behaviors for different types of characters. Theory about artificial intelligence, XNA-framework and Windows Phone -software development from the point of game programming are also part of this thesis.

Keywords: Artificial intelligence, XNA, C#, game programming, mobile software development, Windows Phone, Microsoft

Sisältö

Termit ja Käsitteet

.NET Framework	Microsoftin kehittämä ohjelmointimalli ja ohjelmistokomponenttikirjasto, jota Microsoftin VisualStudio.NET-ympäristössä kehitetyt ohjelmistot käyttävät.
Agentti	Agentit ovat peliin kuuluvia, itsenäisesti toimivia hahmoja, esineitä, pelimaailman osia tms.
Algoritmi	Algoritmi on joukko järjestelmällisesti suoritettuja käskyjä tai ohjeita jonkin toiminnon suorittamiseksi.
Animaatio	Animaatiossa sarja nopeasti näytettäviä kuvia luo illuusion liikkeestä. Animaatio toteutetaan kuva kuvalta.
API	Application programming interface tarkoittaa ohjelmointirajapintaa.
C#	C# (C sharp) on Microsoftin kehittämä oliopohjainen ohjelmointikieli.
Debuggaus	Debuggaus on toimenpide, jossa paikallistetaan ohjelmakoodissa olevia virheitä.
DirectX	DirectX on Microsoftin Windows-käyttöjärjestelmälle kehittämä, erityisesti peleihin tarkoitettu ohjelmointirajapinta sovelluksen ja laitteiston välille.
Emulaattori	Emulaattori on tietokoneohjelma tai laitteistolaajennos, joka mahdollistaa ohjelmien ja joskus myös laitteiden käytön muunlaisella laitteella tai käyttöjärjestelmällä kuin mille ne on alun perin tarkoitettu.

Heuristiikka	Heuristiikka on epävirallinen menetelmä ongelmanratkaisuun. Sitä käytetään metodina, joka nopeasti johtaa yleensä riittävän lähelle parasta mahdollista lopputulosta.
Instanssi	Instanssi (esiintymä, ilmentymä) tarkoittaa olio-ohjelmoinnissa luokan edustajaa eli oliota.
Kehitysympäristö	Kehitysympäristö (tai ohjelmointiympäristö) on ohjelma tai joukko ohjelmia, jolla ohjelmoija suunnittelee ja toteuttaa ohjelmistoa.
Ketterä ohjelmistokehitys	Ketterä ohjelmistokehitys on joukko ohjelmistotuotantoprojekteissa käytettäviä menetelmistöjä, joille on yhteistä toimivan ohjelmiston ensisijaisuus. Ketteryys painottaa muun muassa suoraa viestintää, tiimityöskentelyä, tiivistä asiakasyhteistyötä, nopeaa muutoksiin reagointia ja iteratiivista ohjelmistokehitystä.
Käyttöliittymä	Käyttöliittymä on se laitteen, ohjelmiston tai minkä tahansa muun tuotteen osa, jonka kautta käyttäjä käyttää tuotetta. Tietokoneohjelmassa käyttöliittymä on se osa, jonka käyttäjä näkee näytöllä.
Metodi	Metodi eli funktio on luokan tai olion funktiotyyppinen aliohjelma, joka suorittaa tietyn, rajoitetun tehtävän.
Mobiililaite	Mobiililaite on jokin mukana kulkeva tieto- tai viestintätekniikan laite, kuten esimerkiksi matkapuhelin, PDA, kannettava tietokone tai taulutietokone.

Moniajo	Moniajo on laitteen käyttöjärjestelmän ominaisuus, joka mahdollistaa näennäisesti usean ohjelman ajamisen samanaikaisesti.
Muodostin	Muodostin (tai muodostinfunktio) on funktio, joka suoritetaan, kun muodostetaan luokan instanssi.
Ohjelmistokehys	Ohjelmistokehys (tai sovelluskehys) tarkoittaa ohjelmistotuotetta, joka muodostaa rungon sen päälle rakennettavalle sovellukselle. Ohjelmoinnin apuväline.
Päätöksenteko	Päätöksenteko on valitsemista vaihtoehtojen välillä.
Rajapinta	Rajapinta (tai ohjelmointirajapinta) on määritelmä, jonka mukaan ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään. Rajapinnan päätarkoitus on tarjota käyttömahdollisuus yleisimmille toiminnoille.
Sprite	Sprite on 2D-kuva (kaksiulotteinen kuva).
Testitapaus	Testitapaus keskittyy yhteen tiettyyn tilanteeseen tai ehtoon testattavassa järjestelmässä. Se kuvaa, mitä testataan ja miten, ja ohjaa testin suorittajaa.
WP	WP on lyhenne sanoista Windows Phone.
Xbox	Microsoftin kehittämä pelikonsoli.

1 Johdanto

Peliala on maailmalla viihdeteollisuuden nopeimmin kasvava alue. Vuonna 2012 sen globaali arvo oli 67 miljardia dollaria ja suunta on voimakkaassa kasvussa erityisesti lisääntyneen mobiilipelaamisen myötä. Mobiilipelimarkkinoiden osuus koko peliteollisuuden arvosta on tällä hetkellä noin 7,8 miljardia, mutta sen on arvioitu kolminkertaistuvan seuraavien kolmen vuoden aikana. Mobiilipelaamisen kasvun on mahdollistanut mobiililaitteiden teknologian nopea kehittyminen viime vuosien aikana. Pelialan osalta sama suunta on myös Suomessa, sillä peleistä on tullut Suomen taloudellisesti merkittävin kulttuurivientiala. (Tekes 2012; GamesIndustry International 2012.)

Pelin menestymisen kannalta merkittävimpiä tekijöitä ovat itse pelin pelattavuus sekä siinä esiintyvä tekoäly. Peleissä tekoälyä käytetään hahmojen ohjaamiseen ja älykkään käyttäytymisen luomiseen. Ennen kaikkea sen tarkoitus on tuoda pelaajalle haasteita, elävöittää ja monipuolistaa pelimaailmaa sekä tehdä pelistä viihdyttävä.

Tarkasteltaessa pelialaa yleisesti, laitteistojen, kuten pelikonsoleiden ja PC:iden, kehittyminen on mahdollistanut monimutkaisempien pelien kehittämisen, mikä puolestaan asettaa suuremmat vaatimukset tekoälylle. Näiden vaatimusten täyttäminen mobiilipeleissä on haastavaa ja joskus jopa mahdotonta, sillä huolimatta mobiililaitteiden valtavasta kehityksestä, niiden resurssit ovat yhä melko rajalliset. Mobiililaiteeksi katsotaan kuuluvaksi kaikki mukana kulkevat laitteet perinteisistä matkapuhelimista ja PDA-laitteista aina kannettaviin tietokoneisiin. Puhuttaessa mobiililaitteista tässä opinnäytetyössä tarkoitetaan älypuhelimia ja taulutietokoneita.

Tämän opinnäytetyön tavoitteena on suunnitella ja toteuttaa tekoäly Corrupted Cultura -peliin. Corrupted Cultura on Lappeenrannassa toimivan pelialan yrityksen, Jontkan, ensimmäinen mobiilipeli ja se on suunnattu Windows Phone -käyttöjärjestelmille. Corrupted Culturassa tekoäly esiintyy vihollishahmojen liikkumisessa ja päätöksenteossa.

Opinnäytetyön alussa kerrotaan Jontkasta ja Corrupted Culturasta sekä kuvataan sovelluskehitysprojektin kulkua ja siihen liittyviä vaiheita. Tämän jälkeen luvussa 4 tutustutaan Windows Phone -käyttöjärjestelmän ja Windows Phone -sovelluskehityksen erityispiirteisiin, jotka on otettava huomioon pelikehityksessä. Luku 5 puolestaan keskittyy tekoälyyn ja sen määrittelyyn sekä pelitekoälyn kehittymiseen ja yleisimpiin toteutustekniikoihin. Luku 6 käsittelee XNA-peliohjelmointia ja erityisesti XNA-ohjelmistokehyksen hyödyntämistä pelikehityksessä. Luvut 7, 8 ja 9 muodostuvat Corrupted Culturan tekoälyn suunnitteluun ja toteutukseen liittyvistä teknisistä ratkaisuista. Ne sisältävät muun muassa kuvauksen kehitysympäristöstä, tekoälyn keskeisimpien ominaisuuksien toteutukseen liittyvät ratkaisut ja tekoälyn testaamisen.

2 Jontka ja Corrupted Cultura

Jontka on tammikuussa 2013 perustettu lappeenrantalainen pelialan yritys, jonka yhtiömuotona on osuuskunta. Jontkaan kuuluu yhdeksän eri toimenkuvan omaava osakasta, jotka kaikki ovat vasta valmistuneita tai opintojensa loppusuoralla olevia alansa taitajia. Jontka tekee tiivistä yhteistyötä lähialueen korkeakoulujen kanssa ja tälläkin hetkellä yritys tarjoaa harjoittelua, projektitöitä ja opinnäytetyöaiheita Saimaan ammattikorkeakoulun ja Lappeenrannan teknillisen yliopiston opiskelijoille. Yrityksen toimitilat löytyvät Lappeenrannan teknillisen yliopiston kampukselle rakennetusta pelikehitykseen tarkoitetusta LevelUp-toimintaympäristöstä. LevelUp Lappeenranta on osa Playahubin yhteistyöverkostoa ja se tarjoaa laitteet, ohjelmistot sekä modernit tilat nousevan yrityksen tuotekehitysprojekteille.

Playa on alun perin Kaakkois-Suomessa perustettu verkosto, joka keskittyy pelialan kehittämiseen. Sen keskeisenä ideana on tukea alalla toimivia kansainväliseen liiketoimintaan tähtääviä yrityksiä tarjoamalla pelikehitysympäristön lisäksi myös liiketoimintaa kehittäviä palveluja, osaamista sekä verkostoitumismahdollisuuksia niin Suomessa kuin ulkomaillakin. Playan verkostossa on mukana pelialan yrityksiä, alan toimijoita sekä oppilaitoksia. (Lappeenranta Business & Innovations 2012.)

LevelUp-toimintaympäristön lisäksi Jontka on hakenut avustusta AppCampus- ja BizSpark-hankkeista. AppCampus on Aalto-yliopiston, Nokian ja Microsoftin yhteisprojekti, jonka tavoitteena on luoda mobiilialan yritysten uusi sukupolvi ja tuottaa laadukkaita sovelluksia sekä Windows Phone -laitteille että muille Nokian alustoille. Microsoft ja Nokia tukevat ohjelman toimintaa molemmat yhdeksällä miljoonalla eurolla. Startup-yritys voi saada 20 000, 50 000 tai 70 000 euroa sovelluskehitykseen, riippuen sovelluksen monimutkaisuudesta. BizSpark-ohjelma puolestaan on yksin Microsoftin hanke ja se tarjoaa aloitteleville ohjelmistoyrityksille tai ohjelmistoon perustuville palveluyrityksille lisenssit kehitystyökaluihin ja -alustoihin. Lisäksi se tuo mahdollisuuden verkostoitua rahoittajiin, hautomoihin, järjestöihin ja muihin aloitteleviin yrityksiin. (AppCampus 2013; Microsoft BizSpark 2013.)

Jontka työstää tällä hetkellä kahta peliä, Corrupted Culturaa ja TimeRollia. Työn alla on myös SuomiMoba-webportaali sekä HTML5-toteutuksen Facebook-aplikaatio. Pääpaino on kuitenkin Windows Phone -alustalle suunnatun Corrupted Culturan kehittämisessä.

Corrupted Cultura

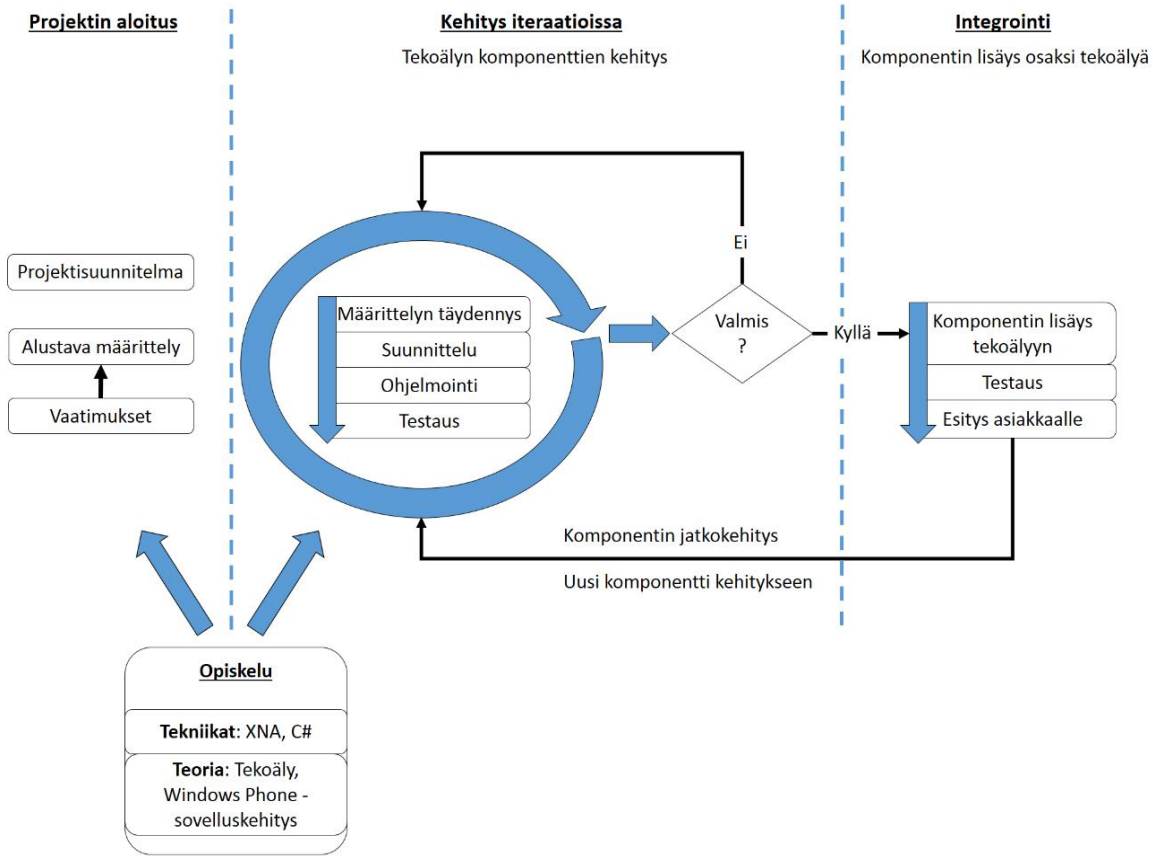
Corrupted Cultura on toiminta-genren peli, jossa taustatarina etenee sarjakuvamaisesti eteenpäin. Lähitulevaisuuteen sijoittuvassa tarinassa pelaajan ohjaama hahmo kulkee halki raunioituneen kaupungin haluten kostaa entisille työnantajilleen, jotka pettivät tämän. Matkaa ovat hidastamassa erilaiset viruksen saastuttamat viholliset, joiden kanssa pelaaja joutuu käymään taisteluja. Peliin on lisätty strategisia elementtejä antamalla pelaajalle mahdollisuus asettaa vihollisille ansoja ennen taisteluiden alkua. Pelaaja pystyy myös siirtelemään pelimaailmasta löytyviä esineitä tehden niistä vihollisten kulkua rajoittavia esteitä. Corrupted Cultura sisältää mustaa huumoria, jota on tuotu esille paitsi juonessa, mutta myös muun muassa vihollishahmoissa ja pelaajan ohjaaman hahmon ominaisuuksissa. (Jontka 2013.)

Corrupted Cultura on suunnattu kaikenikäisille mobiilipelaajille. Alustana toimivat ensisijaisesti Windows Phone -käyttöjärjestelmillä varustetut älypuhelimet, mutta tulevaisuudessa se käännetään mahdollisesti myös muille mobiililaitteille.

Jontka on kehittänyt Corrupted Culturaa ketterän ohjelmistokehityksen periaatteiden mukaan. Tämä näkyi kaikkien yrityksen työntekijöiden yhteistyönä ja tiimityöskentelynä samassa tilassa. Lopullista toteutusta ei myöskään ollut suunniteltu tarkasti, vaan muutoksia määrittelyssä tapahtui kehityksen myötä. Määrittelyä täydennettiin viikoittain, jolloin tarkastettiin myös mahdolliset muutostarpeet. Koko Corrupted Cultura on jaettu kolmeen, erilläänkin toimivaan osaan. Kukin osa tullaan valmistumisen myötä julkaisemaan yksitellen.

3 Projektin eteneminen

Tekoälyn kehittäminen tapahtui ketterälle ohjelmistokehitykselle tyypillisiä piirteitä hyödyntäen, mikä mahdollisti kehityksen melko tiukallakin aikataululla. Pääpaino oli ohjelmoinnissa ja nopeassa suunnittelussa kokonaisvaltaisen dokumentoinnin sijaan. Työ toteutettiin sovelluskehitysprojektina, mutta sen läpivientiin ei käytetty suoraan valmista mallia, vaan eteneminen tapahtui kuvassa 1 esitetyn kulkukaavion mukaan.



Kuva 1. Sovelluskehitysprojektin vaiheet

Projektin etenemisessä oli kolme vaihekokonaisuutta: projektin aloitus, tekoälyn komponenttien iteratiivinen kehitys ja iteraatioissa syntyneiden komponenttien integraatio osaksi tekoälyä. Näistä projektin aloitus oli oma kokonaisuutensa, kun taas iteratiivinen kehitys ja integraatio muodostivat jatkuvan syklin limittyen osittain päällekkäin.

Projektin aloitus

Projektin alussa laadittiin projektisuunnitelma, jossa selvitettiin projektille tavoitteet ja tehtävät sekä rajattiin tehtävän työn määrä. Projektisuunnitelmassa määriteltiin myös projektin aikataulu sekä karkea kuvaus tässä luvussa esitetystä projektin etenemisestä.

Projektin aloitusvaiheeseen kuului myös tekoälyn alustava määrittely asiakkaalta saatujen vaatimusten ja toivomusten perusteella. Tarkkaa suunnitelmaa toteutuksesta ei tässä vaiheessa tehty, vaan määrittelyn odotettiin täydentyvän ja muuttuvan projektin edetessä.

Komponenttien kehitys iteraatioissa ja integraatio tekoälyyn

Tekoälyyn liittyvien komponenttien eli luokkien ja luokkaryhmien kehitys tapahtui lyhyissä iteraatioissa, jotka kestivät tehtävän työn määrästä riippuen viikosta kahteen viikkoon. Jokainen iteraatio oli oma kokonaisuutensa sisältäen määrittelyn täydentämisen, suunnittelun, ohjelmoinnin ja testauksen. Iteraatiota toistettiin, kunnes kehitettävä komponentti täytti määrittelyn ja läpäisi testauksen.

Kun iteraation tuloksena saatu komponentti todettiin valmiiksi, siirryttiin integraatiovaiheeseen, jossa komponentti liitettiin osaksi tekoälyä. Tämän jälkeen testattiin komponentin toiminta muiden komponenttien kanssa ja lopulta esitettiin uusia ominaisuuksia sisältävä tekoäly asiakkaalle. Mikäli komponentti ei läpäissyt testausta tai asiakas ei ollut tyytyväinen, palattiin takaisin iteraatiovaiheeseen. Muuten arvioitiin uudelleen projektin prioriteetit ja päätettiin seuraavan iteraarion sisällöstä.

4 Windows Phone

Windows Phone on Microsoft Corporation kehittämä mobiilikäyttöjärjestelmä, joka on suunnattu älypuhelimille. Vuonna 2010 julkaistu ensimmäinen Windows Phone -käyttöjärjestelmä kulki nimellä Windows Phone 7 ja se oli Microsoftin Windows Mobile -sarjan seuraaja. Vuonna 2011 käyttöjärjestelmä oli mahdollista päivittää versioon 7.5, jota kutsuttiin myös nimellä ”Mango”. Vuotta myöhemmin julkaistiin ”Tango”-päivitys, jonka tehtävänä oli korjata ohjelmointivirheitä ja laskea laitteistovaatimuksia.

Vuonna 2012 Microsoft toi markkinoille kokonaan uuden sukupolven Windows Phone -käyttöjärjestelmän, Windows Phone 8:n. Samalla vanhemman sukupolven laitteet saivat korvaavan 7.8-päivityksen, joka toi osan uuden käyttöjärjestelmän ominaisuuksista käyttöön myös vanhemmille laitteille. Merkittävimmät Windows Phone -laitteiden valmistajat ovat Nokia Oyj, Samsung Group ja HTC Corporation. Näistä suurin on Nokia, joka käyttää Windows Phone 8 -käyttöjärjestelmää älypuhelimiensa ensisijaisena käyttöjärjestelmänä.

Tässä luvussa käsitellään peliohjelmoinnin kannalta oleellisia ja huomioitavia Windows Phone -käyttöjärjestelmän ominaisuuksia ja Windows Phone -sovelluskehityksen erityispiirteitä. Erityisen tärkeää peliohjelmoinnissa, ja Windows Phone -sovelluskehityksessä yleensäkin, on sovelluksen elinkaaren ymmärtäminen.

4.1 Käyttöliittymä

Windows Phone -käyttöliittymän runkona toimii Microsoftin Metro-suunnittelukehys. Metron muotokieltä ja käyttöliittymäkehystä Microsoft on käyttänyt myös Windows 8 -käyttöjärjestelmässä.

Windows Phone -käyttöjärjestelmän aloitusnäyttö (kuva 2) koostuu erilaisista Tile-ruuduista, joita käyttäjä pystyy kiinnittämään, poistamaan ja siirtämään haluamallansa tavalla. Windows Phonen 7.8- ja 8-versioissa onnistuu myös ruutujen koon muuttaminen. Tile-ruudut voivat olla esimerkiksi ihmisiä, sovelluksia tai musiikkia. Niiden painaminen käynnistää kyseistä Tileä esittävän sovelluksen. Li-

säksi niin sanotut Live Tile -ruudut voivat esittää käyttäjälle dynaamisesti muuttuvaa informaatiota, minkä ansiosta käyttäjä saa automaattisesti päivityksiä haluamistaan asioista.

Kuva 2. WP 8 käyttöliittymä: aloitusnäyttö – sovellusluettelo – pelit

Aloitusnäytössä esiintyvät vain käyttäjän siihen kiinnittämät sovellukset. Kaikki asennetut sovellukset, pelejä lukuun ottamatta, löytyvät sovellusluettelosta. Pelit näytetään Pelit-toiminnossa, joka sisältää käyttäjän pelikokoelman. Kuvassa 2 on havainnollistettu Windows Phone käyttöliittymää käyttäen Windows Phone 8-emulaattoria.

Windows Phone -käyttöjärjestelmän laitteistovaatimuksissa on, että laitteesta löytyy kuvassa 3 esitetyt toimintopainikkeet: Back, Start ja Search. Laitevalmistajasta riippuen nämä on voitu toteuttaa hipaisupainikkeilla tai fyysisillä näppäimillä.

Kuva 3. WP-käyttöjärjestelmän vaatimat toimintopainikkeet

Sovellukset käyttävät Back-painiketta omassa navigoinnissaan, jolloin sen painaminen vie edelliseen näkymään. Mikäli ollaan sovelluksen ensimmäisellä sivulla,

painikkeen painaminen lopettaa sovelluksen. Start-painike palauttaa käyttäjän käyttöjärjestelmän aloitusnäyttöön. Search-painiketta käyttöjärjestelmä puolestaan käyttää hakutoiminnon aloittamiseen.

4.2 Laitteistovaatimukset

Microsoft edellyttää, että kaikki Windows Phone -käyttöjärjestelmiä käyttävät laitteet täyttävät laitteistolle asetetut minimivaatimukset. Rajoituksilla on pyritty varmistamaan se, että käyttöjärjestelmät toimisivat sujuvasti kaikilla laitteilla. Alla on listattu sekä Windows Phone 7:n että 8:n laitteistovaatimukset. Tiedot perustuvat Engadget www-sivuston artikkeleiden tietoihin.

Windows Phone 7 -laitteiden minimivaatimukset (Ziegler 2010):

- 1GHz Qualcomm Snapdragon suoritin DirectX9 tuella
- kapasitiivinen, vähintään neljän pisteen monikosketusnäyttö
- kaksi resoluutiovaihtoehtoa: WVGA (800x480) ja HVGA (480x320)
- vähintään 256 MB RAM-muistia
- 8 GB Flash-muisti
- näppäimet: Start, Back, Search, kamera, virta/lukitus, äänenvoimakkuuden lisäys/vähennys
- 5 megapikselin kamera LED-salamalla
- kiihtyvyysanturi, avustettu GPS-vastaanotin, valontunnistin

Windows Phone 8 -laitteiden minimivaatimukset (Molen 2012):

- Qualcomm Snapdragon S4 kaksoisydinsuoritin
- kapasitiivinen, vähintään neljän pisteen monikosketusnäyttö
- kolme resoluutiovaihtoehtoa: WXGA (1280x768), 720p (1280x720) ja WVGA (800x480)
- DirectX9 yhteensopiva grafiikkasuoritin, jossa tuki Direct3D kiidytykselle
- WVGA-laitteille vähintään 512 MB RAM-muistia, 720p ja WXGA laitteille vähintään 1 GB RAM-muistia
- 4 GB Flash-muisti
- näppäimet: Start, Back, Search, kamera, virta/lukitus, äänenvoimakkuuden lisäys/vähennys

- takakamera LED- tai Xenon-salamalla
- kiihtyvyysanturi, valontunnistin, etäisyystunnistin, värinämoottori, GPS, (lisäksi magnetometri ja gyroskooppi vaihtoehtoisina)

4.3 Erityispiirteitä Windows Phone -sovelluskehityksessä

Kaikki Windows Phone sovellukset on kirjoitettu .NET-pohjaisilla ohjelmointikielillä. Tämän opinnäytetyön koodiesimerkit ja liitteenä olevan Corrupted Cultura-pelin vihollishahmojen tekoälyn ohjelmakoodi on kirjoitettu C#-kielellä. Windows Phone -sovelluksia olisi kuitenkin mahdollista kirjoittaa myös käyttäen esimerkiksi Visual Basic .NET -ohjelmointikieltä.

4.3.1 Silverlight ja XNA

Windows Phone tarjoaa kehittäjille kaksi erityyppistä sovellusalustaa: Silverlight ja XNA Framework. Nämä sovellusalustat jakavat keskenään joitain luokkakirjastoja, mikä mahdollistaa joidenkin XNA-kirjastojen käyttämisen Silverlight-ohjelmissa ja toisinpäin. Sama ohjelma ei kuitenkaan voi sekoittaa molempien alustoiden visuaalisuuksia. Yleisesti Silverlightia käytetään kirjoitettaessa niin sanottuja ”tavallisia sovelluksia” ja työkaluja. (Petzold 2010, 3.)

XNA:ta puolestaan käytetään kirjoitettaessa tehokkaita ja vaativaa grafiikkaa sisältäviä pelejä. Se tarjoaa työkalut 2D-pelien hahmojen ja taustojen luontiin sekä 3D-pelien mallien määrittelyyn 3D-avaruudessa. Itse pelien toimita, joka sisältää graafisten kohteiden liikuttelun näytöllä sekä reagoinnin käyttäjän syötteeseen, on synkronoitu sisäänrakennetulla XNA-pelisilmukalla. (Petzold 2010, 3.)

Sekä Silverlightin että XNA:n ajonaikaiset kirjastot löytyvät valmiina Windows Phone -käyttöjärjestelmistä, joten käyttäjän ei itse tarvitse asentaa niitä (Dawes 2010, 4). XNA:han perehdytään tarkemmin luvussa 6.

4.3.2 Sovelluksen elinkaari

Windows Phone -käyttöjärjestelmässä sovellusten moniajo on rajoitettua. Käyttöjärjestelmä pystyy suorittamaan taustalla useita käyttöjärjestelmään kuuluvia järjestelmäsovelluksia, mutta kolmansien osapuolten sovelluksien moniajo ei ole sallittua. Käytännössä tämä tarkoittaa sitä, että vain yksi sovellus voi olla kerralla

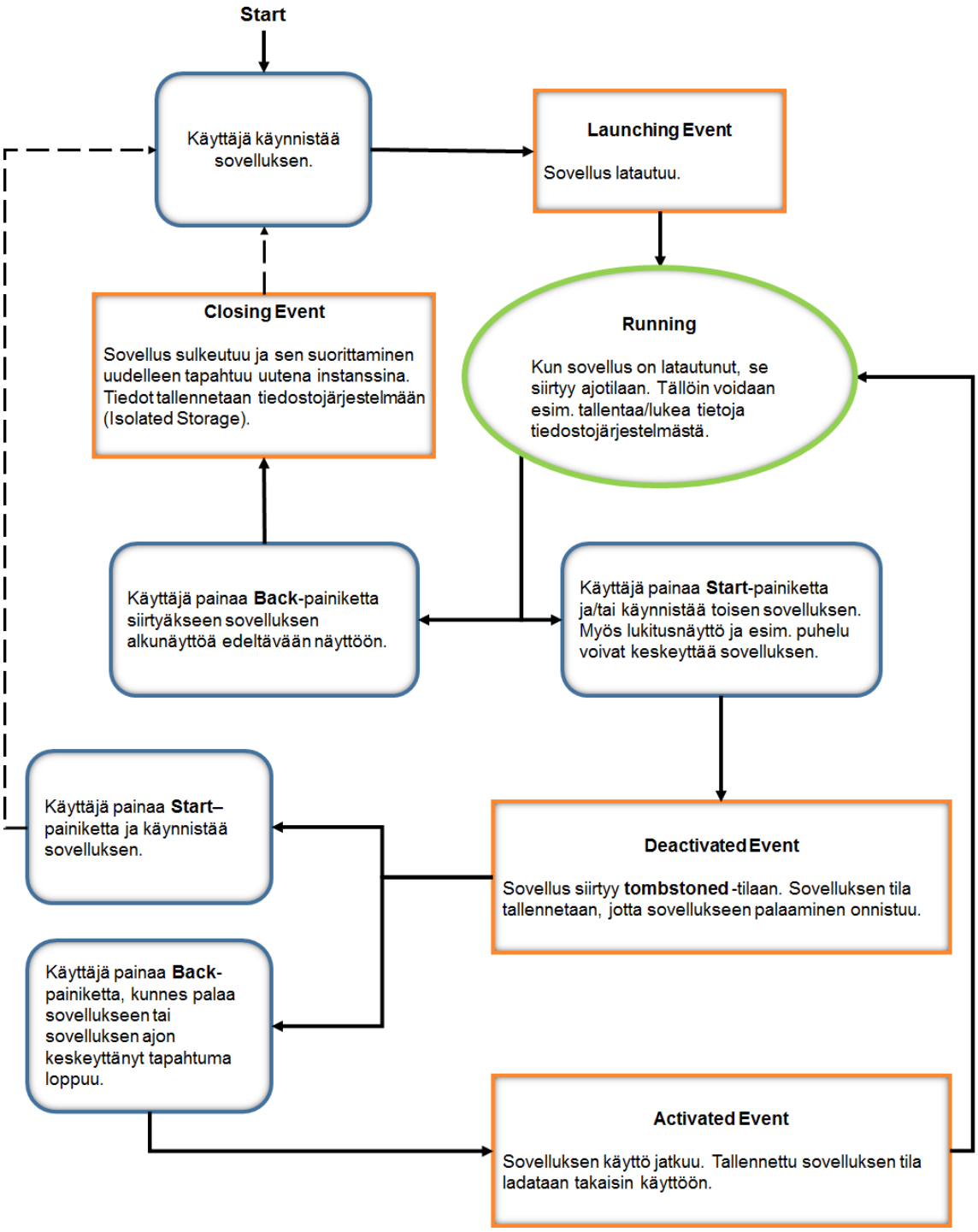
aktiivisena. Kun käyttäjä siirtyy pois sovelluksesta, käyttöjärjestelmä sulkee sen. Sama tapahtuu myös silloin, kun jokin tapahtuma (event), kuten puhelu tai kalenterin muistutus keskeyttää ohjelman suorittamisen. Suunniteltaessa ja toteutettaessa sovelluksia Windows Phonelle, tämä ominaisuus on otettava huomioon ja varmistettava, että ohjelma pystyy sekä tallentamaan että palauttamaan tilansa. Toisin sanoen ohjelman käyttöä on pystyttävä jatkamaan siitä, mihin se on keskeytynyt.

Tähän Windows Phone tarjoaa niin sanotun tombstoning-tekniikan, jonka avulla käyttäjä ei välttämättä edes huomaa, että sovellus on ollut suljettuna. Käyttöjärjestelmä itsessään ei käytä tombstoning-tekniikka kolmansien osapuolien sovelluksille, vaan tilan tallennus ja palautus on sovelluskehittäjien hoidettavana.

Suoritusmalli ja tilanhallinta

Windows Phonen suoritusmallli (execution model) määrää sovelluksen elinkaaren sen käynnistämisestä aina lopettamiseen asti. Suoritusmalli on suunniteltu tuomaan loppukäyttäjälle madollisimman nopea ja herkästi reagoiva käyttöympäristö. Aktiivisten sovellusten rajoittaminen yhteen poistaa mahdollisuuden, että käyttäjä saa laitteensa tilaan, jossa useat sovellukset kilpailevat samoista rajoitetuista resursseista. Tällainen tila aiheuttaisi muun muassa suorituskyvyn heikkenemistä ja akun nopeaa kulumista. (Microsoft Developer Network 2013.)

Kuvassa 4 on esitetty sovelluksen elinkaari Windows Phone -käyttöjärjestelmässä. Kuva perustuu Microsoftin MSDN-sivustolla (Microsoft Developer Network) julkaisemaan ”Application Lifecycle”-oppaaseen ja siinä esiintyvään kaavioon. Kyseinen opas on suunnattu Windows Phone -sovelluskehittäjille ja se antaa yksityiskohtaiset ohjeet sovelluksen elinkaaren ja tilan hallintaan.



Kuva 4. Sovelluksen elinkaari WP-käyttöjärjestelmässä

Launching ja Running

Sovelluksen elinkaari alkaa sen käynnistämisestä, jolloin luodaan instanssi sovelluksesta. Samalla käynnistyy *Launching*-tapahtuma, jonka aikana luetaan tiedostojärjestelmästä vain kaikki välttämättömät tiedot sovelluksen näyttämiseksi.

Kun Launching-tapahtuma on käsitelty, sovellus siirtyy *Running*-tilaan, joka vastaa normaalia sovelluksen ajoa. (Microsoft Developer Network 2013.)

Käytettäessä XNA-sovelluskehystä Launching-tapahtumaan siirrytään vasta Initialize- ja LoadContent-metodien kutsumisen jälkeen (Dawes 2010, 335). Sen seurauksena näiden metodien alustamia tietoja on mahdollista käyttää jo Launching-tapahtuman koodissa. XNA:han sekä edellä mainittuihin metodeihin perehdytään tarkemmin tämän opinnäytetyön luvussa 6.

Closing

Running-tilaa seuraava tila riippuu käyttäjän tekemistä valinnoista. Jos käyttäjä painaa Back-painiketta ollessaan sovelluksen ensimmäisellä sivulla, siirrytään *Closing*-tapahtumaan ja ohjelman suoritus loppuu. Closing-tapahtuman käsittelyn yhteydessä ohjelman täytyy tallentaa sen tiedostojärjestelmään kaikki ajon aikaiset tiedot, joiden halutaan säilyvän eri sovellusinstanssien välillä. (Microsoft Developer Network 2013.)

Deactivated ja tombstone-tila

Jos käyttäjä siirtyy pois sovelluksesta Start-painikkeella, sovelluksen elinkaaressa siirrytään *Deactivated*-tapahtumaan. Tähän päädytään myös, jos ajossa oleva sovellus korvataan toisella sovelluksella tai tapahtumalla, kuten saapuvalla puhelulla. Mikäli halutaan mahdollistaa sovellukseen palaaminen ja toiminnan jatkaminen siitä, mihin se keskeytyi, täytyy sovelluksen tila tallentaa. Tilan tallennukseen on kaksi tapaa. Ensimmäinen tallettamistapa on pysyville tiedoille (persistent data), joiden tallennus tiedostojärjestelmään hoidetaan sovelluksen ohjelmakoodissa. Tätä tapaa täytyy käyttää kaikille ajonaikaisille tiedoille, jotka halutaan myöhemmin palauttaa. Toinen tallennustapa on väliaikaisille tiedoille (transient data), jotka tiedostojärjestelmän sijaan tallennetaan PhoneApplicationService-luokan State-ominaisuuteen. (Dawes 2010, 335–337.)

Tilatietojen tallennus sovelluksesta pois siirryttäessä ja niiden palautus sovellukseen palatessa muodostavat menettelytavan, jota kutsutaan termillä *tombstoning*. Kun Deactivated-tapahtuma on käsitelty ja sovellus on tallentanut tilatiedot,

puhutaan, että sovellus on tällöin *tombstone*-tilassa (Microsoft Developer Network 2013).

Activated

Kun käyttäjä palaa tombstone-tilassa olevaan sovellukseen tapahtuu Activated-tapahtuman suoritus. Tässä tapahtumassa sovelluksen täytyy lukea tiedot PhoneApplicationService-luokan State-ominaisuudesta ja palautua tilaan jossa se oli ennen kuin siitä siirryttiin pois. Sovelluksen pitää myös palauttaa kaikki tarpeelliset tiedot tiedostojärjestelmästä. (Microsoft Developer Network 2013.)

4.3.3 Isolated Storage

Jokaisella Windows Phone -sovelluksella on oma eristetty tilansa, Isolated Storage, käyttöjärjestelmän tiedostojärjestelmässä. Sovelluksella ei ole suoraa pääsyä käyttöjärjestelmän tiedostojärjestelmään eikä myöskään muiden sovellusten tiedostojärjestelmiin. Tämä parantaa järjestelmän turvallisuutta, estää luvattoman pääsyn tiedostoihin sekä vähentää tiedostojen korruptoitumista. (Microsoft Developer Network 2012.)

4.4 Dev Center ja Windows Phone Store

Sovellusten testaus onnistuu käyttäen emulaattoria, mutta jossain vaiheessa kehitteillä oleva sovellus täytyy pystyä asentamaan, suorittamaan ja debuggamaan myös fyysisellä Windows Phone -laitteella. Tämä vaatii liittymisen Windows Phone Dev Centeriin ja puhelimen rekisteröimisen kehityslaitteeksi.

Dev Centeriin rekisteröityminen tapahtuu Microsoft-tilillä (Windows Live ID) ja sen vuosittainen jäsenmaksu on 99 dollaria eli noin 77 euroa (Microsoft Developer Network 2013). Kun kehittäjä on maksanut kyseisen maksun, voi hän rekisteröidä kolme Windows Phone -laitetta kehitystarkoitukseen. Laitteiden lukituksen poistaminen onnistuu käyttämällä Windows Phone Developer Registration -työkalua (kuva 5), joka sisältyy Windows Phone -kehittäjätyökaluihin. Kehittäjätyökaluista kerrotaan luvussa 7.2.

Kuva 5. Puhelimen rekisteröinti kehityslaitteeksi

Dev Center -jäsenyys tuo myös pääsyn Windows Phone Storeen (aiemmin Windows Phone Marketplace), joka on Windows Phone -sovellusten kauppapaikka. Kyseinen sovelluskauppa on niin sanottu suljettu sovelluskauppa eli kehittäjien ainut keino julkaista sovelluksiaan. Ennen julkaisua jokaisen sovelluksen on läpäistävä sertifikaattivaatimukset, jotka on kehittäjiä varten listattu Dev Centerissä. Tällä menettelyllä Microsoft takaa sovelluksien laadukkuuden ja turvallisuuden. (Microsoft Developer Network 2013.)

5 Tekoäly

Tämä luku keskittyy käsittelemään peleissä esiintyvää tekoälyä sekä tekoälyä yleisellä tasolla ja vertailemaan näiden eroja. Pelitekoälyn osalta kerrotaan sen kehittymisestä peleissä sekä listataan siltä yleisimmin vaadittuja ominaisuuksia. Lisäksi nimetään keskeisimpiä pelitekoälyyn liittyviä toteutustekniikoita.

5.1Tekoälyn määritelmä

Tekoälyn tarkoituksena on saada laitteet ja sovellukset tuottamaan ihmismäistä käyttäytymistä ja toimintoja, joihin elävät olennot kykenevät. Tietokoneiden alkuajoista saakka on ollut monenlaisia ohjelmia, joita on käytetty ratkaisemaan ongelmia, jotka ovat ihmiselle mahdottomia ratkaista. Monia tällaisista ongelmista,

kuten esimerkiksi aritmetiikkaa sekä aineistoon liittyvää lajittelua ja hakua, on alun perin tarkasteltu tekoälyä vaativina ongelmina. Kun ne on myöhemmin saatu ratkaistua kattavammin, ovat ne myöskin erkaantuneet tekoälyn kehittäjien osa-alueelta. (Millington & Funge 2009, 4.)

Tietokoneet ovat hyviä laskennassa, mutta monet ihmiselle yksinkertaiset asiat ovat niille vaikeita. Tällaisia asioita ovat muun muassa kasvojen tunnistaminen, oman puheen tuottaminen, päätöksenteko, ympäristön tulkinta, tulevaisuuden arviointi ja luovuus. Nämä ominaisuudet lasketaan nykykäsityksen mukaan tekoälyksi. Tekoälyä kehitettäessä pyritään nimenomaan selvittämään, millaisilla tekniikoilla ja algoritmeilla edellä mainitut ominaisuudet pystytään esittämään. (Millington & Funge 2009, 4.)

5.2 Akateeminen tekoäly

Akateemisesti tekoälyä tutkitaan kolmesta eri näkökulmasta: filosofisesta, psykologisesta ja insinöörimäisestä. Filosofinen näkökulma pyrkii ymmärtämään ajattelun ja älykkyyden luonteen sekä rakentamaan ohjelmia mallintamaan ajatusten toimintaa. Psykologisesta näkökulmasta kiinnostuneet keskittyvät ihmisaivojen mekaniikan ja mielen prosessien hahmottamiseen ja jäljittelyyn. Tekoälyn tutkiminen insinöörimäisestä näkökulmasta keskittyy suunnittelemaan algoritmeja, jotka jäljittelevät ihmismäisiä toimintoja ja tehtäviä. Pelikehittäjät ovat pääsääntöisesti kiinnostuneet tekoälystä vain insinööritaitona: tehdään algoritmeja, joilla pelihahmot saadaan näyttämään eläviltä ja ajattelevilta olennoilta. (Millington & Funge 2009, 4-7.)

Millington ja Funge jakavat akateemisen tekoälyn karkeasti kolmeen, osittain päällekkäiseen aikakauteen: varhaiseen, symboliseen ja moderniin aikakauteen. Varhainen aikakausi käsittää ajan jo ennen tietokoneita, jolloin filosofit alkoivat lähestyä tekoälyä erilaisten kysymysten kautta. Pohdittuja kysymyksiä olivat muun muassa ”Voiko elottomalle kohteelle antaa elämän?” ja ”Mikä aiheuttaa ajatuksen?”. Toisen maailmansodan aikana, 1940-luvulla, tarve murtaa vihollisten salakirjoitusta ja suorittaa vaativaa laskentaa ydinsotaa varten johtivat ensim-

mäisten ohjelmoitavien tietokoneiden kehitykseen. Kun tietokoneet alkoivat selvitä ihmiselle mahdottomista laskutoimituksista, ohjelmoijat alkoivat kiinnostua tekoälystä ja sen tuomista mahdollisuuksista. (Millington & Funge 2009, 5.)

1950-luvun lopulta aina 1980-luvun alkuun tekoälytutkimuksen pääpaino oli ”symbolisissa” järjestelmissä. Symbolisessa järjestelmässä algoritmi on jaettu kahdeksi komponentiksi: tietojoukoksi ja sitä hallinnoivaksi päättelyalgoritmiksi. Tietojoukko koostuu symboleista, kuten sanoista, numeroista, lauseista tai kuvista, joista päättelyalgoritmi luo uusia yhdistelmiä, jotka esittävät ratkaisuja ongelmiin tai tuovat uutta tietoa. Nykyään symbolista lähestymistapaa käytetään esimerkiksi peleissä, jotka sisältävät liitutauluarkkitehtuuria (blackboard architecture), polunhakua (pathfinding), päätöspuita (decision tree), tilakoneita (state machine) tai ohjausalgoritmeja (steering algorithms). (Millington & Funge 2009, 5.)

Symbolisille järjestelmille tyypillinen ominaisuus on kompromissin tekeminen tietojoukon koon ja päättelyalgoritmin tekemän työn välillä. Tätä Millington ja Funge kutsuvat tekoälyn kultaiseksi säännöksi: Mitä enemmän tietoa on, sitä vähemmän työtä vastauksen löytämiseksi täytyy tehdä. Toisaalta, mitä enemmän työtä voidaan tehdä, sitä vähemmän tietoa tarvitaan. (Millington & Funge 2009, 5-6.)

Lähestyttäessä 1990-lukua symbolinen lähestymistapa alkoi käydä riittämättömäksi kahdesta syystä. Ensinnäkin se oli kehittäjien näkökulmasta liian yksiulotteinen, eikä pystynyt hoitamaan todellisen maailman monimutkaisuutta. Toiseksi se ei kyennyt tarjoamaan filosofisesta näkökulmasta tarkasteltuna riittävän biologisesti uskottavaa tekoälyä. Nämä tekijät ohjasivat tekoälyn kehitystä kohti luonnollista laskentaa (natural computing), jossa tekniikat saivat vaikutteita biologiasta tai muista luonnon systeemeistä. Nämä tekniikat sisältävät muun muassa neuroverkot, geneettiset algoritmit ja simuloidun jäähdytyksen. (Millington & Funge 2009, 6.)

Nykyisin käytännön tekoälytutkimuksen tavoitteet ja tulokset ovat jääneet alan alkuaikoihin verrattuna vaatimattomammiksi. Vaikka tietokoneiden laskentatehot ovat lisääntyneet, tekoäly ei ole juurikaan kehittynyt. Se ei ole kymmeniin vuosiin päässyt yhtään lähemmäs ihmismäistä älykkyyttä ja kyvykkyyttä. Syynä tähän on

se, että käyttökelpoinen teoria älykkyyden mekaanisesta toteuttamisesta puuttuu yhä. Kehityksen pysähtyminen on johtanut jopa itse tekoälyn määritelmän latistumiseen – joissain yhteyksissä tekoäly katsotaan vain osaksi algoritmiikkaa. (Kokkarinen 2003, 296.)

5.3 Pelitekoäly

Jotta pystytään tekemään tulosta pelitekoälyn kehityksessä, täytyy ymmärtää edellä kuvattua akateemista tekoälyä ja siihen liittyviä tekniikoita. Toisaalta erot näiden kahden päämärissä ovat huomattavat. Siinä missä akateeminen tekoäly tavoittelee esimerkiksi ihmistajunnan jäljittelyä, pelitekoäly keskittyy olemaan viihdyttävä ja hauska (Rabin 2002, 9).

Kirby (2011) määrittelee pelitekoälyn hahmon kyvyksi toimia älykkäästi muuttuvissa olosuhteissa. Tämä määritelmä edellyttää, että tekoälyn tuottamat toiminnot ovat pelaajan havaittavissa. Mikäli pelaaja ei havaitse toimintoja, ovat ne turhia tekoälyn kannalta ja täten laskentatehon tuhlausta. Lisäksi toimintojen täytyy olla älykkäitä pelin sisältö huomioiden. Pelaaja on lopulta se, joka määrittelee ”älykkyyden” näiden toimintojen perusteella. Tekoälyn reagointi muuttuviin olosuhteisiin tekee pelistä pelattavan ja luo vuorovaikutuksen pelaajan ja pelin välillä – pelaajan tekemät valinnat vaikuttavat pelin tilaan ja etenemiseen. (Kirby 2011, 1-3.)

Rabinin (2002) mukaan pelitekoälyä ei pitäisi kutsua ollenkaan sanalla ”tekoäly”, vaan parempi nimi sille olisi ”agenttisuunnittelu” tai ”käyttäytymismallinnus”. Syy tähän on se, että sana ”älykkyys” jo itsessään on monimutkainen ja vaikeasti määriteltävissä. Lisäksi pelitekoälyn suunnittelussa on huomioitava annettu konteksti, jossa agenteilla ei välttämättä tarvitse olla ihmismäistä älykkyyttä. Joskus sitä ei ole edes haluttu olevan. Jokaisella pelillä on oma ainutlaatuinen kontekstinsa, jonka puitteissa myös pelin tekoälyä tulisi kehittää. Onnistuneimmat pelitekoälyt ovatkin saaneet alkunsa, kun kehittäjät ovat selvästi tunnistaneet tietyn pelin ongelmat ja miettineet niihin yksilölliset ja ainutkertaiset ratkaisut. (Rabin 2002, 9.)

5.4 Pelien tekoälyn kehittyminen

Pelien tekoäly sai alkunsa peliteollisuuden alkuaikoina 1970-luvulla. Ensimmäisenä tekoälynsä saivat yksinkertaiset kolikkopelit, joissa tekoälyn tärkeimpänä tehtävä oli varmistaa, että pelaaja jatkaa rahan syöttämistä koneeseen pelaamisen jatkamiseksi. 1970- ja 1980-lukuvut olivat tekoälyn kannalta mullistavaa aikaa, sillä silloin peleissä alkoi esiintyä yksinkertaisia sääntöjä ja tapahtumasarjoja, joihin oli onnistuttu lisäämään sattumanvaraista päätöksentekoa tekemään käyttäytymisestä ennalta arvaamatonta. Tuon ajan tunnetuimmat pelit, kuten Pong, Pac-Man ja Donkey Kong, antavat vaikutteita vielä tämän päivän tekoälyihin. Erityisesti Pac-Manin tekoälyn merkitys koko pelitekoälyn historiassa oli merkittävä. Kyseisen pelin tekoäly oli ensimmäinen, jossa oli pelaajaa vastaan toimivat vihollishahmot, jotka seurasivat pelaajan liikkeitä ja tekivät pelaamisesta haastavaa. (Rabin 2002, 3; Millington & Funge 2009, 7.)

Perinteisten kolikkopelien seuraajana olivat erilaiset strategiapelit, joita toteutettiin aluksi lautapelien pohjalta. Shakki on esimerkki klassisesta strategiapelistä, jonka haastavan ja monimutkaisen tekoälyn luomiseen on käytetty paljon tutkimustyötä. Strategiapelin hyvä pelattavuus vaatii toimivan tekoälyn, eivätkä grafiikka ja muut ominaisuudet ole niin suuressa roolissa. Tekoälyn täytyy suorittaa monimutkaisia pelihahmokohtaisia toimintoja sekä erittäin korkeatasoisia strategisia ja taktisia agenttien toimintoja. (Rabin 2002, 3-4.)

Peliteollisuuden alkupuolella grafiikan renderöinti vei niin paljon prosessorin laskentatehoa, ettei tekoälyn vaatimiin algoritmeihin ja laskentoihin ollut enää tarpeeksi resursseja. Laitteiston kehittyessä myös pelien tekoäly on kehittynyt koko ajan monipuolisemmaksi ja viihdyttävämmäksi (Rabin 2002, 3). Tältäkin osin pelitekoäly eroaa akateemisesta tekoälystä. Kuten jo aiemmin mainittiin, akateemisen tekoälyn kehitys on hidastunut laskentatehon lisääntymisestä huolimatta. Pelitekoäly on pystynyt kehittymään, koska se ei edes yritä päästä ihmismäiseen älykkyyteen.

Joidenkin pelien kohdalla tekoälyn rinnalle on kehittynyt myös käsite ”keinoelämästä” (artifical life, A-Life). Tästä esimerkkinä The Sims -pelisarjan pelit, jotka

ovat tunnettuja tekoälyagenttien persoonallisuuden syvyydestä. Keinoelämä pyrkii kehittämään simuloituja hahmoja sekä rakentamaan älykkäästi toimivia hahmoja jonkinlaisen evoluutioprosessin tuloksena. Tällöin pelimaailman tapahtumat ja ominaisuudet määräävät sen miten hahmot kehittyvät ja käyttäytyvät. (Rabin 2002, 3-5; Kokkarinen 2003, 292.)

Rabinin (2002) mukaan tekoälyn merkitys itse pelin teon onnistumisen ja sen menestymisen kannalta on ollut 2000-luvun alusta asti koko ajan suuremmassa osassa. Tämän takia myös videopelien kehittäjät ovat alkaneet ottaa tekoälyn kehittämisen vakavasti sen sijaan että toteuttavat sen viime hetken kiireessä. Pelikehitys on saanut tekoälyyn erikoistuneita ohjelmoijia, jotka ovat mukana peliprojekteissa alusta asti. Monissa tapauksissa jopa ohjelmoijat ilman aikaisempaa kokemusta tekoälystä ovat onnistuneet tuottamaan korkealaatuisen pelitekoälyn. Tämä osoittaa, että tekoälyn kehitys ei välttämättä vaadi erityisosaamista, vaan usein riittää todellisuudentaju, pieni luovuus ja riittävä aika työn loppuun tekemiseksi. (Rabin 2002, 3-5.)

5.5 Pelitekoälyn jaottelu ja yleisimmät toteutustekniikat

Pelin tekoälylle on asetettu useimmiten kolme perusvaatimusta: mahdollisuus liikuttaa peliahmoja, mahdollisuus päättää, miten ja mihin liikutaan, ja mahdollisuus ajatella taktisesti tai strategisesti. Näiden lisäksi joissain tapauksissa vaatimuksena on myös pelihahmon oppiminen ja pelimaailmaan liittyvä tekoäly. Seuraavassa on kuvattu lyhyesti jokainen vaatimuksista ja esitetty yleisimmät toteutustekniikat, joilla ne voidaan saavuttaa.

5.5.1 Liikkumistekoäly

Pelitekoälyn keskeisimpänä vaatimuksena on liikuttaa pelin hahmoja järkevästi. Perustan hahmojen liikuttamiselle luo hahmon sijainti sekä mahdolliset muut ominaisuudet liikkeen hallinnoimiseen. Näitä tietoja voidaan hahmon liikuttamiseksi hyödyntää erilaisissa algoritmeissa yhdessä pelimaailmasta saatavien tietojen kanssa. Pelimaailmasta käytettäviä tietoja voivat olla esimerkiksi seinät, alueen rajat, esteet ja muut hahmot, joilla niilläkin on omat sijaintinsa ja ominaisuutensa. Liikkumiseen käytettävään tekoälyyn liittyy usein animaatioiden käyttö, sillä ne ovat myös osa liikettä. Vaikka animaatiot eivät varsinaisesti kuulukaan tekoälyn

piiriin, on niiden osuus hahmon liikettä toteutettaessa huomioitava. (Lecky-Thompson 2008, 29; Millington & Funge 2009, 39-41.)

Käytetyimmät tekniikat liikkumistekoälyn luomiseen ovat kinemaattiset ja dynaamiset liikkumisalgoritmit sekä polunhakualgoritmit. Kinemaattiset algoritmit käyttävät hahmon staattisia tietoja (sijainti ja suunta) ja lopputuloksena antavat hahmolle nopeuden, jonka suunta on kohteeseen. Dynaamiset liikkumisalgoritmit puolestaan huomioivat lisäksi myös hahmolla jo olemassa olevan liikkeen eli sen nopeuden. Siinä missä kinemaattinen algoritmi antaa hahmolle vakionopeuden kohteeseen, dynaaminen algoritmi osaa myös esimerkiksi vähentää nopeuden suuruutta lähestyttäessä kohdetta. Esimerkkejä dynaamisista algoritmeista ovat Seek, Flee, Arrive, Wander, Collison Avoidance ja Obstacle Avoidance. (Millington & Funge 2009, 40-95.)

Kineettisten ja dynaamisten liikkumisalgoritmien suurimpana ongelmana on se, etteivät ne osaa laskea tarvittavaa liikettä etukäteen. Hahmo, joka käyttää pelkästään näitä algoritmeja liikkumistekoälyssään, reagoi esimerkiksi edessä olevaan esteeseen vasta, kun on törmännyt tai törmäämässä siihen. Tämä johtaa helposti ”ei älykkääseen”-toimintaan, kuten jumiutumiseen. Kyseisiin ongelmiin ratkaisun tarjoavat erilaiset polunhaku-algoritmit, jotka osaavat etsiä hahmolle edullisimman reitin kohteeseen. Yleisimmin käytettyjä polunhaku-algoritmeja ovat Dijkstra- ja A*-algoritmit sekä niihin pohjautuvat algoritmit. (Millington & Funge 2009, 204-237.)

5.5.2 Päätöksenteko

Päätöksenteko määrää sen, mitä hahmo tekee seuraavaksi. Yleensä pelitekoälyissä jokaisella hahmolla on joukko erilaisia käyttäytymismalleja, joista valitaan se mitä suoritetaan. Tyypillisiä käyttäytymismalleja ovat hyökkää, puolustaudu, piiloudu, vartioi, etsi ja niin edelleen. Päätöksenteossa tekoäly selvittää, mikä käyttäytymismalleista on milläkin hetkellä kaikkein tarkoituksenmukaisin. Valittu käytös suoritetaan hyödyntäen liikkumistekoälyä ja animaatioita. Käytetyimmät tekniikat päätöksentekoon ovat tilakoneet ja päätöspuut. (Millington & Funge 2009, 293-295.)

Tilakoneen tarkoitus on jakaa tekoäly ja hahmon toiminta pieniin ja helposti hallittaviin osiin eli tiloihin. Tällaisia tiloja ovat esimerkiksi hyökkäys, puolustautuminen, piiloutuminen, vartiointi ja niin edelleen. Tilojen lisäksi tilakone sisältää keinot tilojen välillä siirtymiseen ja kussakin tilassa suoritettavat toiminnot. Tilakoneen toiminnan mallintamiseen käytetään tilakaavioita. (Kirby 2011, 43-47.)

Käytännössä tilakoneen tila määrittää, mitä hahmo on tekemässä kyseisellä ajanhetkellä. Varsinainen päätöksenteko esiintyy silloin, kun hahmo siirtyy tilasta toiseen suorittaen jotain tilan määräämää käyttäytymismallia. Esimerkki tilakoneen toteutuksesta on esitetty luvussa 8.8.

5.5.3 Taktinen ja strateginen tekoäly

Monissa peleissä liikkuminen ja päätöksenteko riittävät luomaan viihdyttävän ja toimivan tekoälyn. Kaikissa niihin liittyvissä tekniikoissa on kuitenkin kaksi merkittävää rajoitusta: ne on tarkoitettu käytettäväksi vain yhdelle hahmolle, ja ne eivät osaa päätellä saatavilla olevista tiedoista ennustetta tilanteiden kehittymiselle. Tätä varten on olemassa taktinen ja strateginen tekoäly. Tähän kategoriaan kuuluvat tekoälyalgoritmit, jotka eivät ohjaa vain yhtä hahmoa, vaan vaikuttavat kokonaisten hahmojoukkojen käyttäytymiseen. Jokaisella joukon hahmolla voi olla omat päätöksenteko- ja liikkumisalgoritmit, mutta valitut päätökset vaikuttavat koko joukon toimintaan. (Millington & Funge 2009, 10.)

5.5.4 Oppiminen ja ympäristön tekoäly

Oppiminen on pelitekoälyn osa-alue, joka mahdollistaa jatkuvasti kehittyvien ja muuttuvien pelihahmojen luomisen. Sen avulla saadaan uskottavampia hahmoja ja pelikohtainen tekoäly: hahmot pystyvät oppimaan ympäristöstään ja ominaisuuksista, joita itse tuottavat sekä käyttämään oppimaansa parhaan tuloksen saavuttamiseksi. (Millington & Funge 2009, 579.)

Ympäristön tekoäly puolestaan ei liity pelihahmoihin tai pelaajaan, vaan ympäröivään pelimaailmaan. Pääasiallisesti ympäristön tekoäly käsittelee pelaajan toimintojen vaikutusta pelimaailman muutoksiin. (Lecky-Thompson 2008, 31.)

6 XNA

XNA on Microsoftin kehittämä, oliopohjainen peliohjelmointiin tarkoitettu työkalu. Se on rakennettu DirectX:n päälle ja pyritty tekemään mahdollisimman yksinkertaiseksi ja helppokäyttöiseksi ohjelmoijia varten. Nykyään XNA ei ole pelkkä ohjelmistokehys tai luokkakirjasto, kuten esimerkiksi DirectX-rajapinta, vaan se sisältää paljon työkaluja ja jopa oman Visual Studioon integroituvan ohjelmointiympäristön. XNA on täysin ilmainen ja se tarjoaa kehittäjille mahdollisuuden tuottaa yhtäaikaisesti pelejä Windowsille, Xbox 360 -pelikonsolille ja Windows Phonelle.

6.1 Johdatus XNA-peliohjelmointiin

Microsoft aloitti XNA:n kehittämisen 2000-luvun alussa ja ensimmäinen versio siitä julkaistiin alkuvuodesta 2006 nimellä ”XNA Build March 2006 CTP”. XNA Build on työkalu monimutkaisten pelien projektinhallintaan ja käännöstyöhön. Seuraava askel XNA:n kehityksessä oli XNA rajapinnan ja ohjelmointiympäristö XNA Game Studio Expressin ensimmäisen beta-version julkaisu elokuussa 2006. Tuolloin monet sovelluskehittäjät ja harrastajat kokeilivat XNA:ta ja kirjoittivat nopeasti monia pieniä 2D-pelejä hyödyntäen XNA:n Sprite-luokkia. Tässä vaiheessa XNA ei vielä sisältänyt juurikaan 3D-toiminnallisuuksia ja omien 3D-grafiikkaa sisältävien pelien kirjoittaminen oli vaikeaa. Ensisijaisesti XNA Game Studio Express oli suunnattu aloittelijoille, harrastajille ja opiskelijoille tarjoten heille mahdollisuuden tuottaa nopeasti omia pelejä Windows-käyttöjärjestelmille sekä Xbox 360 -pelikonsoleille. (Nitschke 2007, 4.)

Versio 2.0 sisälsi tuen verkkopelien kehittämiselle Xbox Liven kautta. XNA-pelikehitys käyttäen kaikkia Visual Studio 2005 versioita oli myös mahdollista. 3.0-version julkaisun myötä tuli tuki Zunelle ja mahdollisuus pelien julkaisuun ja myyntiin Xbox Livellä Xbox Live Community Gamesin (nykyinen Xbox Live Indie Games) kautta. (Miller & Johnson 2011, 3.)

XNA Game Studio 4.0 on tällä hetkellä uusin versio, jossa merkittävimpänä uutena ominaisuutena ovat työkalut Windows Phone 7:n kehitykselle. Muita päivityksiä ovat muun muassa yksinkertaisempi grafiikka API ja muut ominaisuudet, kuten mikrofoonituki ja dynaaminen ääni. (Miller & Johnson 2011, 3.)

6.2 XNA-ohjelmistokehys

XNA-ohjelmistokehyksen (XNA Framework) ja XNA Game Studion kehittäminen on kulkenut alusta asti rinnakkain ja tällä hetkellä molemmat ovat versiossa 4.0. XNA-ohjelmistokehys on kokoelma pelien kehitykseen suunniteltuja kirjastoja, jotka perustuvat Microsoftin .NET Framework 4:ään. Niiden tarkoituksena on vähentää peleissä yleisesti käytettävien komponenttien uudelleen kirjoittamisen tarvetta. Lisäksi ohjelmistokehys pyrkii piilottamaan laitteistokohtaiset yksityiskohdat ja eroavaisuudet, mikä helpottaa ohjelmointityötä ja lisää ohjelmakoodin yhteensopivuutta eri alustoilla.

6.2.1 Luokkakirjastot

XNA-ohjelmistokehykseen kuuluu luokkia, palveluita, rajapintoja ja tietotyyppejä sisältäviä kirjastoja. Itse ohjelmistokehys on osana XNA Game Studiota. Nämä kirjastot tarjoavat pääsyn XNA-ohjelmistokehyksen toiminnallisuuksiin, ja ne on suunniteltu perustaksi XNA Game Studio -ohjelmien, -komponenttien ja -ohjainten rakentamiselle. Microsoftin MSDN-sivustolla (Microsoft Developer Network) on luettavissa dokumentaatio XNA-ohjelmistokehyksen luokkakirjastojen nimiavaruuksista ja tarkat kuvaukset niiden sisältämien luokkien ominaisuuksista. Kyseisen dokumentaation pohjalta nimiavaruudet ja niiden luokat on listattu pääpiirteittäin taulukossa 1.

Nimi	Selitys
Microsoft.Xna.Framework	Tarjoaa useimmiten käytetyt luokat, kuten ajastimet ja pelisilmukat.
Microsoft.Xna.Framework.Audio	Sisältää matalan tason ohjelmointirajapinnan (API) metodit, joilla käytetään ja muokataan XACT:lla tuotettuja projekteja ja sisältöä äänen toistamiseen.
Microsoft.Xna.Framework.Content	Sisältää ajonaikaiset sisältöputken (Content Pipeline) komponentit.
Microsoft.Xna.Framework.Design	Tuo yhtenäisen tavan muuntaa tietotyyppejä toisiksi tietotyypeiksi.
Microsoft.Xna.Framework.GamerServices	Pitää sisällään luokkia, jotka toteuttavat erilaisia pelaajiin liittyviä palveluita. Palvelut kommunikoivat suoraan pelaajan tai pelaajan datan kanssa tai

	muuten heijastaa pelaajan valintoja. Se sisältää myös syöttölaitteiden ja profiilidatan ohjelmointirajapinnat.
Microsoft.Xna.Framework.Graphics	Sisältää matalan tason ohjelmointirajapinnan metodit, jotka hyödyntävät grafiikkaprosessoria 3D kohteiden näyttämiseksi.
Microsoft.Xna.Framework.Graphics.PackedVector	Esittää tietotyyppejä komponenteilla, jotka eivät ole 8-bitin kerrannaisia.
Microsoft.Xna.Framework.Input	Sisältää luokat syötteen vastaanottamiseen näppäimistöltä, hiirestä ja Xbox 360 –peliohjaimelta.
Microsoft.Xna.Framework.Input.Touch	Sisältää luokat, jotka sallivat pääsyn kosketusperäiseen syötteeseen laitteilla, jotka sitä tukevat.
Microsoft.Xna.Framework.Media	Sisältää luokat musiikkikappaleiden, soittolistojen, albumeiden ja kuvien listaamiseen, toistamiseen ja näyttämiseen.
Microsoft.Xna.Framework.Net	Pitää sisällään luokat, jotka antavat tuen Xbox LIVE:lle, moninpelille ja verkkoyhteyksille.
Microsoft.Xna.Framework.Storage	Sisältää luokat, jotka sallivat tiedostojen lukemisen ja kirjoittamisen.

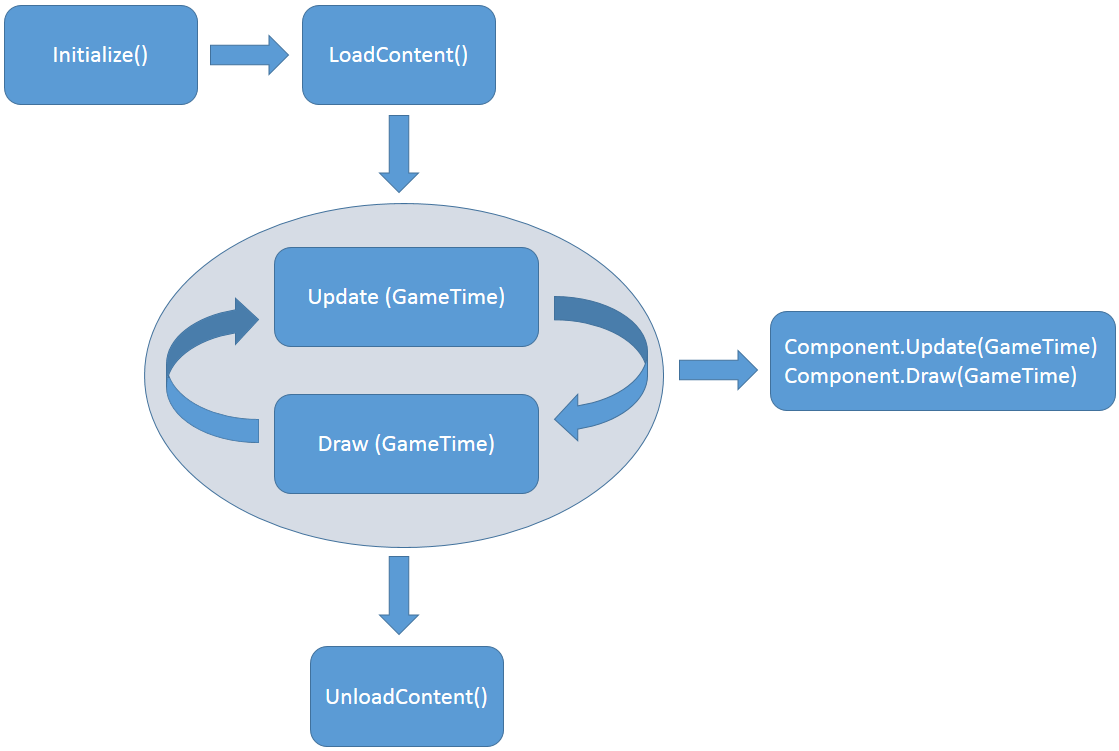
Taulukko 1. XNA framework 4.0

6.2.2 Pelisilmukka

XNA-ohjelmistokehys (Microsoft.Xna.Framework) sisältää Game-luokan, joka toteuttaa game loopin eli pelisilmukan. Pelisilmukka tarjoaa pelille sekä ikkunan, jossa se näytetään että ylikuormitettavat metodit (kuva 6), jotka helpottavat pelin ja käyttöjärjestelmän välistä kommunikointia. Käytännössä tämä näkyy kehittäjälle siten, että pelisilmukka automatisoi pelilogiikan päivityksen ja sisällön uudelleen piirtämisen. (Microsoft Developer Network 2013.)

XNA:lla toteutettu peli vaatii pääluokan, joka periytyy Game-luokasta. Kyseisessä luokassa täytyy ylikirjoittaa Initialize-, LoadContent-, UnloadContent-, Update- ja Draw-metodit sekä suorittaa kaikki peliin liittyvä toiminnallisuus näistä käsin.

Game-luokka metodeineen muodostaa siis rungon, jonka päälle varsinaista peliä lähdetään kehittämään.



Kuva 6. Pelin pääluokan pelisilmukka

Initialize()

Kun XNA-peli suoritetaan, käydään ensiksi pääluokan muodostimen ohjelmakoodi läpi. Tämän jälkeen kutsutaan Initialize-metodia. Kyseisen metodin tehtävänä on alustaa peli ennen sen varsinaista näyttämistä, ja se on hyvä paikka esimerkiksi resoluution tai näytön suunnan asettamiselle. Initialize-metodin kutsuminen tapahtuu vain kerran pelin aikana. (Jaegers 2010, 16.)

LoadContent()

Kun alustus on suoritettu, kutsutaan LoadContent-metodia. Tätä metodia käytetään kaikkien grafiikka- ja äänilähteiden lukemiseen. Metodissa alustetaan myös spriteBatch-olio, jota käytetään kohteiden piirtämiseen Draw-metodissa. (Jaegers 2010, 16.)

Update()

LoadContent-metodista XNA-peli siirtyy loputtomaan silmukkaan, joka kutsuu Update-metodia tietyin väliajoin (Jaegers 2010, 17). Se, kuinka tiheään kutsuminen tapahtuu, on esitetty kohdassa ”Eteneminen pelisilmukassa”. Update-metodi on vastuussa koko pelilogiikan hallinnasta ja päivittämisestä. Sen kautta tapahtuu käyttäjän syötteen rekisteröinti, spritejen liikuttaminen, pisteiden laskeminen, tekoälyn päivittäminen ja niin edelleen kaikki muu paitsi kohteiden piirtäminen.

Update-metodi saa parametrina gameTimen, jota pelisilmukka käyttää määrittämään kuluneen ajan edellisestä Update-kutsusta. Jos Update-metodin sisältämän pelilogiikan päivittäminen kestää liian kauan, pelisilmukassa hypätään Draw-metodin yli (Jaegers 2010, 16-17). Pääpaino on siis ensisijaisesti Update-metodin suorittamisessa, mikä näkyy pelaajalle pelin ”nykimisenä” silloin, kun Draw-metodi jää suorittamatta. Tästä ilmiöstä kerrotaan tarkemmin suorituskyvyn resursseja käsittelevässä luvussa 9.1.

Draw()

Draw-metodi hoitaa kuvaruutujen piirtämisen. Se on siis vastuussa siitä, että sen hetkinen pelitila esitetään näytöllä pelaajalle. Normaalisti tätä metodia kutsutaan aina jokaisen Update-metodikutsun jälkeen.

UnloadContent()

UnloadContent()-metodia kutsutaan, kun varattuja resursseja täytyy vapauttaa. Esimerkiksi olion tuhoamisen yhteydessä on varmistettava, että kaikki olion varaamat resurssit vapautetaan takaisin käyttöön.

Eteneminen pelisilmukassa

Pelisilmukassa eteneminen voi tapahtua joko vakiosiirtymällä (fixed-step) tai muuttujasiirtymällä (variable-step). Siirtymä tyypin valintaan vaikuttaa se, kuinka usein Update-metodia täytyy kutsua. Toisin sanoen siirtymän määrää se, miten aikaan sidotut tapahtumat kuten liike ja animaatiot halutaan esittää. Vakiosiirtymässä Game-luokka yrittää kutsua Update-metodiaan TargetElapsedTime-muuttujaan perustuvilla vakio-aikaväleillä. Oletuksena fixed-step pelisulmukassa

metodi-kutsu tapahtuu 1/60 sekunnin välein. Jos peli käyttää muuttujasiirtymää, tapahtuu Update- ja Draw-metodien kutsuminen jatkuvassa silmukassa riippumatta TargetElapsedTime-muuttujan arvosta. Variable-step-pelisilmukan aloitus tapahtuu asettamalla Game-luokan muuttujan IsFixedTimeStep arvoksi false. (Microsoft Developer Network 2013.)

Windows Phonessa metodi-kutsut tapahtuvat 30 kertaa sekunnissa. Tämän seurauksena käyttöjärjestelmän suurin kuvataajuus on 30 kuvaa sekunnissa ja sen asettaminen tapahtuu pelin pääluokan muodostimessa kuvan 7 mukaisella ohjelmakoodilla.

```
// Windows Phone Frame rate oletuksena 30fps
TargetElapsedTime = TimeSpan.FromTicks(333333);
```

Kuva 7. Kuvataajuuden asettaminen

6.2.3 Pelikomponentit

Pelikomponentit tarjoavat modulaarisen tavan lisätä peliin toiminnallisuuksia. Pelikomponentti luodaan periyttämällä uusi luokka joko GameComponent-luokasta tai DrawableGameComponent-luokasta. Jälkimmäistä käytetään silloin, kun komponentin täytyy pystyä lataamaan ja piirtämään graafista sisältöä. Pelilogiikan ja piirto-ominaisuuksien lisääminen pelikomponenttiin tapahtuu ylikirjoittamalla perityn kantaluokan Update-, Draw- ja Initialize-metodit. Näitä metodeja kutsutaan pääluokan eli Game-luokan vastaavien metodien kutsumisen yhteydessä. (Microsoft Developer Network 2013.)

Kuvan 8 ohjelmakoodiesimerkissä on havainnollistettu, kuinka Game-luokan Draw-metodissa kutsutaan DrawableGameComponent-luokasta periytettyjen ”hahmo” ja ”este” -luokkien instanssien Draw-metodeja. Käytännössä kaikki yksittäiset kohteet piirretään tämän metodin kautta, joten se on vastuussa kaikesta näytöllä esitettävästä sisällöstä. Samalla periaatteella tapahtuu myös pelikomponenttien päivitys eli Update-metodien suoritus.

```
/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    hahmo.Draw(gameTime);
    este.Draw(gameTime);

    base.Draw(gameTime);
}
```

Kuva 8. Komponentin piirto Draw-metodikutsulla

Vaihtoehtoisesti pelikomponentti-luokkien instanssit voidaan lisätä pelin pääluokan Components-kokoelmaan, kuten kuvassa 9 on esitetty. Tällöin pelisilmukka hoitaa automaattisesti kaikkien pelikomponenttien päivittämisen ja piirtämisen. Tätä opinnäytetyötä tehdessä käytettiin kyseistä menetelmää, koska se helpotti luvussa 8.9 kuvatun piirtojärjestyksen hallintaa.

```
hahmo = new Hahmo(this);
este = new Este(this);
this.Components.Add(hahmo);
this.Components.Add(este);
```

Kuva 9. Olioiden lisäys pääluokan Components-kokoelmaan

6.2.4 Pelipalvelut

Pelipalvelut puolestaan ovat mekanismi, jolla olioiden välisiä vuorovaikutuksia voidaan ylläpitää. Pelipalvelut toimivat niin sanotun välittäjän (mediator) kautta, joka on Game.Services-luokka. Palvelun tuottajat rekisteröidään välittäjälle, jolta palvelua tarvitsevat palvelun käyttäjät voivat pyytää tarvitsemaansa palvelua. Tämä mahdollistaa palvelun pyytämisen ilman, että tarvitaan suoraa viittausta jokaiseen palvelua tarjoavaan olioon. (Microsoft Developer Network 2013.)

7 Kehitysympäristö ja käytetyt tekniikat

Tekoälyn ohjelmointityö tapahtui olio-ohjelmoinnin periaatteiden mukaan käyttäen C#-ohjelmointikieltä ja edellisessä luvussa kuvattua XNA-ohjelmistoke-

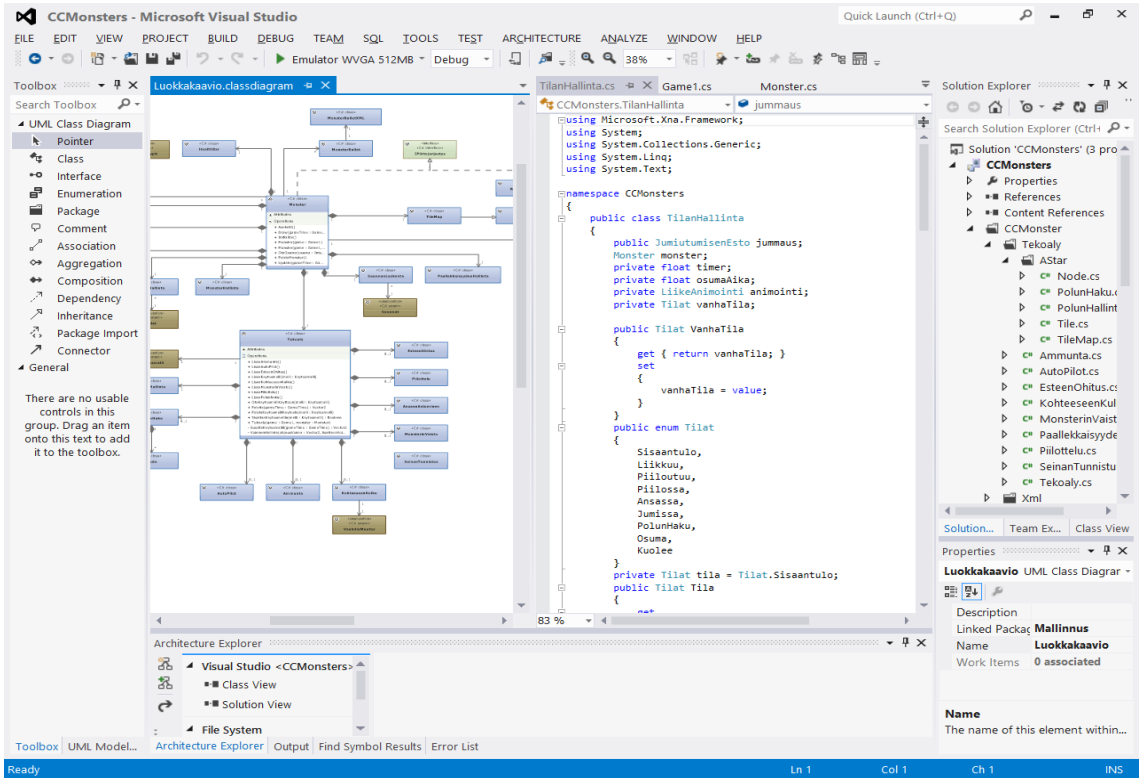
hystä. C# on Microsoftin kehittämä oliopohjainen ohjelmointikieli, joka on suunniteltu .NET Framework -ohjelmistokomponenttikirjastoa käyttävien sovellusten tekoon.

Tekoälyn suunnittelussa ja toteutuksessa käytettävän kehitysympäristön muodostivat Visual Studio 2012 ja siihen integroidut laajennokset. Tässä luvussa on kuvattu lyhyesti kehitysympäristöön kuuluvat työkalut.

7.1 Microsoft Visual Studio 2012

Tämän opinnäytetyön keskeisimpänä työkaluna käytettiin Microsoft Visual Studio 2012 Ultimate -ohjelmointiympäristöä (IDE, Integrated development environment). Visual Studiolla voidaan tuottaa muun muassa Windows-, web- ja mobiilisovelluksia käyttäen useita ohjelmointikieliä ja niiden yhdistelmiä. Mahdollisia ohjelmointikieliä ovat esimerkiksi Visual Basic, Visual C#, Visual C++, Visual F# ja JavaScript (Microsoft Developer Network 2013). Lisäksi Visual Studion toiminnallisuutta voidaan laajentaa integroimalla siihen erilaisia lisäosia.

Kuvassa 10 on havainnollistettu Visual Studion työskentelynäkymä. Visual Studiota käytettiin ohjelmakoodin kirjoittamisen lisäksi muun muassa luokkakaavion suunnittelussa ja tekoälyn debuggauksessa. Visual Studio oli oleellinen työkalu myös tekoälyn testauksessa.



Kuva 10. Visual Studio 2012 Ultimate

7.2 Windows Phone SDK 8.0

Pelikehitys ja yleisestikin sovelluskehitys Windows Phonelle vaatii Windows Phone SDK -kehitystyökalujen asentamisen ja integroinnin Visual Studioon. SDK:sta käytettiin versiota 8.0, joka mahdollisti pelikehityksen sekä uudelle Windows Phone 8:lle että vanhemmalle Windows Phone 7:lle.

SDK 8.0 asentuu lisäosana Visual Studio 2012 Professional-, Premium- tai Ultimate-versioihin. Vaihtoehtoisesti voi käyttää myös SDK-kehitystyökalupakettiinkin kuuluvaa Express for Windows Phone -versiota, jolla on mahdollista kehittää vain puhelimessa toimivia sovelluksia käyttäen C#- tai Visual Basic -ohjelmointikieliä. Lisäksi SDK 8.0 vaatii vähintään 64-bittisen Windows 8 Pro -käyttöjärjestelmän. (Microsoft 2013.)

SDK 8.0 sisältää muun muassa seuraavat kehitystyökalut:

- Microsoft Visual Studio Express 2012 for Windows Phone
- Windows Phone Emulaattorit
- Microsoft Expression Blend for Windows Phone

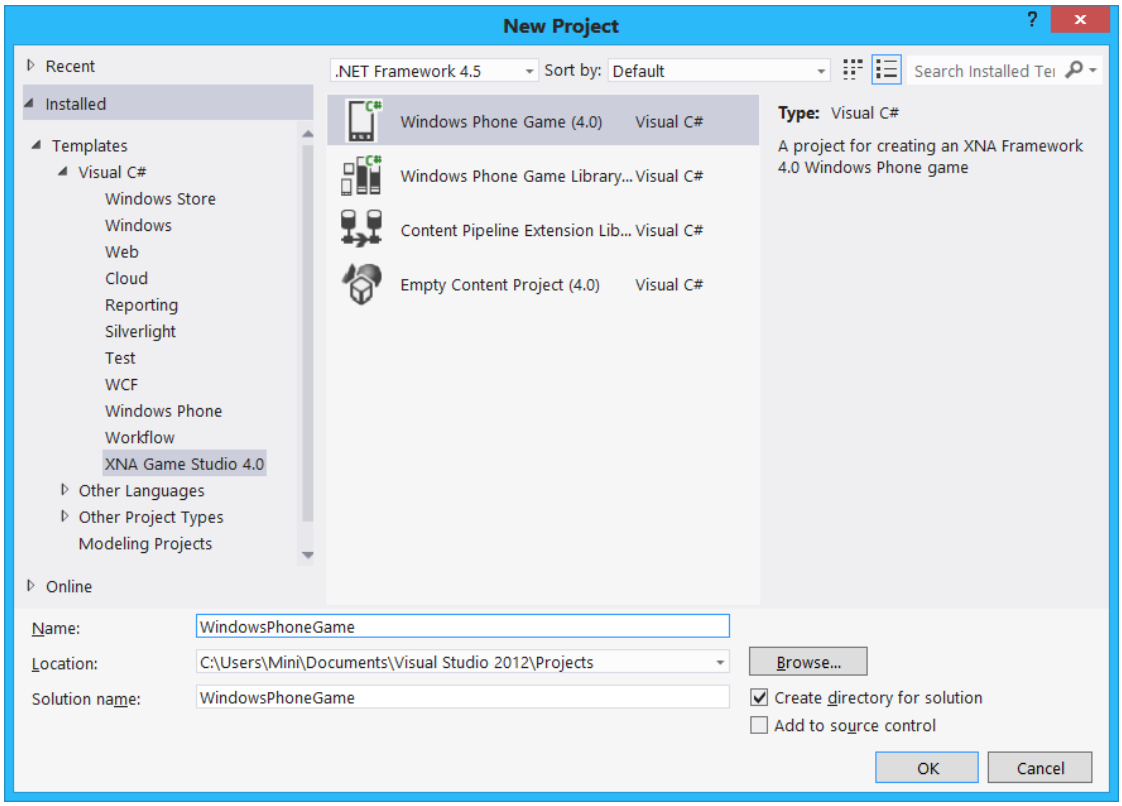
- Microsoft Team Explorer
- XNA Game Studio 4.0

Näistä käytettiin emulaattoria sekä Xna Game Studiota, joista kerrotaan tarkemmin seuraavissa luvuissa. Hyödylliseksi havaittiin myös SDK:n tarjoamat työkalut sovellusten suorituskyvyn monitorointiin ja analysointiin. Niillä pystyttiin testaamaan, kuinka paljon tekoälyssä käytetyt algoritmit käyttävät laitteen resursseja, mikä mahdollisti algoritmien optimoinnin mobiililaitteelle sopivaksi.

7.2.1 XNA Game studio 4.0

XNA Game Studio integroituna Visual Studioon luo ohjelmointiympäristön, joka mahdollistaa pelien tekemisen Windowsille, Xbox 360 -pelikonsolille ja Windows Phonelle. Game Studio sisältää XNA-ohjelmistokehyksen, joka koostuu pelikehitykseen suunnitelluista kirjastoista. XNA-ohjelmistokehyksestä ja sen hyödyntämisestä peliohjelmoinnissa on kerrottu tarkemmin luvussa 6.2. Lisäksi Game Studio sisältää työkalut grafiikka- ja audiosisällön lisäämiseksi peleihin.

Pelin teko Game Studiolla aloitetaan luomalla Visual Studiossa uusi XNA Game Studio 4 Windows Phone Game -projekti, kuten kuvassa 11 on esitetty. Tällöin Visual Studio luo automaattisesti uuden peliprojektin ja rungon, jonka päälle peliä on helppo lähteä rakentamaan. Tämän rungon keskeisin osa on pelisilmukka, jonka toiminta ja käyttö on kuvattu luvussa 6.2.2.

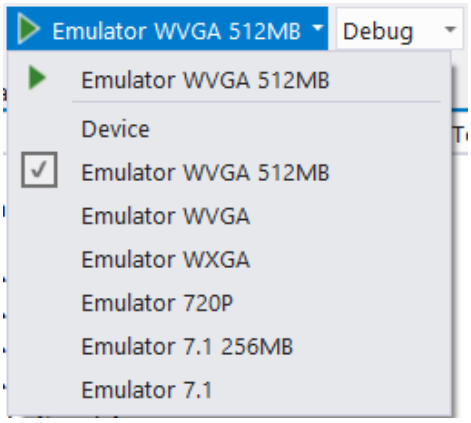


Kuva 11. Uuden peliprojektin luonti

7.2.2 Windows Phone -emulaattori

Windows Phone -emulaattori (kuva 2) tarjoaa virtualisoidun ympäristön, jossa sovellusten debuggaus ja testaus onnistuu ilman fyysistä laitetta. Emulaattori on suunniteltu siten, että se vastaa suorituskyvyltään mahdollisimman tarkasti oikeaa Windows Phone -laitetta.

SDK 8.0 sisältää Windows Phone 8 -emulaattorit 1 GB RAM-muistilla WVGA, WXGA ja 720p resoluutioille sekä 512 MB RAM-muistilla ja WVGA resoluutiolle. Lisäksi käytettävissä on 512 MB ja 256 MB RAM-muisteilla varustetut Windows Phone 7.1 (vastaa versiota 7.5 eli mangoa) -emulaattorit. Emulaattoreilla pystytään siis ajamaan sovelluksia lähes kaikissa luvussa 4.2 esitettyjen laitteistovaatimusten kokoonpanoissa. Emulaattorin käynnistys on esitetty kuvassa 12.



Kuva 12. Emulaattorin käynnistys

Windows Phone 8 emulaattorin ajaminen tapahtuu erillisenä virtuaalikoneena Hyper-V-virtuaaliympäristössä. Emulaattori ja Hyper-V vaativat toimiakseen 64-bittisen Windows 8 Pro -käyttöjärjestelmän lisäksi myös SLAT-yhteensopivan (Second Level Address Translation) prosessorin. Emulaattoreita ja niiden käyttämiä resursseja on mahdollista muuttaa ja hallita Hyper-V Manager -työkalulla, jonka saa käyttöön, kun Hyper-V on sallittu Windowsin ominaisuuksista. Esimerkiksi tätä opinnäytetyötä tehdessä emulaattorin käyttämää RAM-muistin määrää ja prosessoritehoa täytyi vaihdella tekoälyn algoritmien suorituskykyä testatessa.

8 Corrupted Culturan tekoäly

Corrupted Culturassa esiintyy viisi erilaista vihollishahmotyyppiä, joista kolme on mukana Corrupted Culturan ensimmäisessä osassa. Tekoälyn vaatimuksena oli, että sitä voidaan käyttää eri tavoin eri hahmotyypeillä. Näin jokainen hahmo käyttäytyisi sen tyypille ominaisella tavalla, mikä elävöittää peliä ja tekee siitä monipuolisemman. Tekoälyn tuli mahdollistaa hahmojen liikkuminen pelimaailmassa ja antaa edellytykset yksinkertaiselle päätöksenteolle. Tekoälylle asetettujen vaatimusten mukaan sen täytyi antaa hahmolle seuraavat ominaisuudet:

- liikkuminen pelimaailmaa hyödyntäen
- kulkeminen kohti pelaajaa
- pelimaailman esteiden tunnistaminen
- piiloutuminen esteen taakse
- hyökkäykset (ammunta ja lähitaistelu)

- ansoihin reagointi
- pelaajan syötteisiin reagointi.

Näitä ominaisuuksia on kutsuttu sekä tässä raportissa että ohjelmakoodissa käyttäytymismalleiksi. Tässä luvussa käydään läpi tekoälyn keskeisimmät ominaisuudet ja niiden toteuttaminen. Ratkaisuja vastaava ohjelmakoodi on nähtävillä tämän opinnäytetyön liitteessä 2. Liitteeseen 2 on koottu myös muita tekoälyn toimivuuden kannalta merkittäviä luokkia, joita ei ole mainittu tässä raportissa.

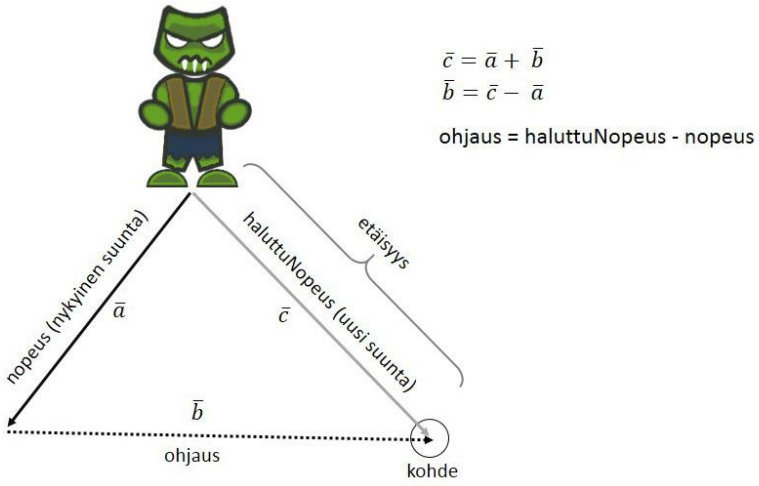
8.1 Luokkakaavio

Projektin aikana toteutetut luokat ja niiden väliset yhteydet on esitetty luokkakaaviossa, joka löytyy tämän opinnäytetyön liitteestä 1. Luokkia on kaiken kaikkiaan 45 ja ne kuuluvat joko tekoälyyn, vihollishahmon hallintaan tai animaatioiden toteutukseen. Liitteen luokkakaavio on pelkistetty malli todellisesta kaaviosta, mikä tekee siitä helppolukuisemman ja helpottaa kokonaisuuden hahmottamista. Kaavion luokkien tiedoista näytetään vain luokan nimi. Lisäksi luokkien välisiä yhteyksiä on piilotettu siten, että esillä ovat vain oleellisimmat. Esimerkiksi kaikilla tekoälyyn liittyvillä luokilla on ominaisuutena viittaus monster-luokan olioon, mutta kaavion selkeänä pitämisen vuoksi ne on piilotettu.

8.2 Kulkeminen kohteeseen

Corrupted Culturan määrittelyn mukaan vihollishahmojen täytyi ensisijaisesti kulkea koko ajan lyhintä mahdollista reittiä pelaajan luo. Tämä onnistuu helposti asettamalla hahmolle nopeus-vektori, jonka suunta on kohti päätepistettä eli kohdetta. Nopeuden suuruus ja suunta eivät kuitenkaan voi pysyä koko ajan samana, koska hahmojen täytyy muun muassa pystyä pysähtymään ja väistämään eteen tulevia esteitä sekä toisia hahmoja. Nopeuden muuttuessa hahmoa täytyy jatkuvasti uudelleenohjata kohti kohdetta.

Uudelleenohjaus toteutettiin käyttäen kuvassa 13 havainnollistettua ohjaus-vektoria. Sen selvittämiseksi täytyy ensin laskea nopeus, jonka hahmo tarvitsee kohteen saavuttamiseksi ideaalitapauksessa (eli silloin, kun edessä ei ole esteitä). Tätä on kutsuttu sekä tässä raportissa että ohjelmakoodissa nimellä haluttuNopeus.



Kuva 13. Ohjaus-vektorin ratkaiseminen

Ohjaus-vektori lisättynä hahmon nykyisen nopeuden vektoriin tuottaa haluttuNopeus-vektorin. Kun haluttuNopeus-vektori ja nopeus-vektori tiedetään, voidaan ohjaus-vektori ratkaista näiden erotuksena kuvassa 13 esitetyllä tavalla. Uudelleenohjauksen laskentaa täytyy suorittaa niin kauan, kun hahmon ja kohteen välinen etäisyys on suurempi kuin nolla.

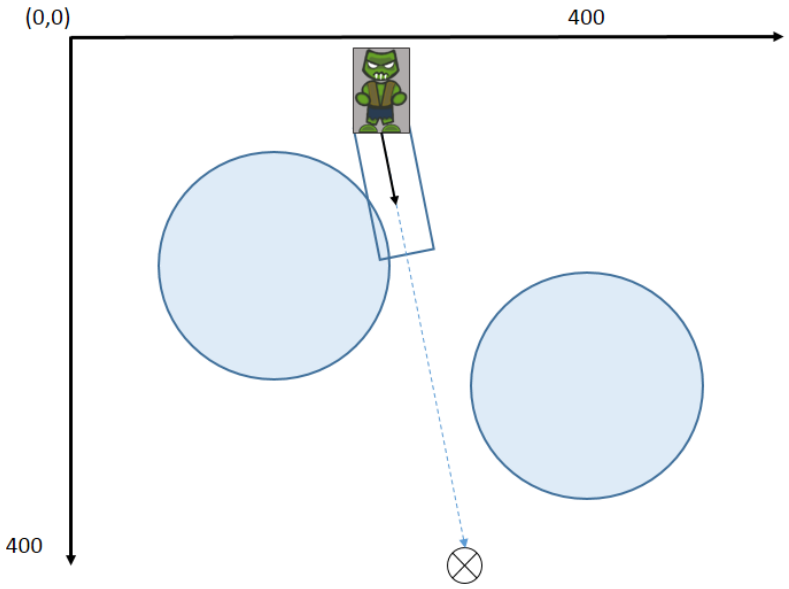
Luonnollisuutta hahmon liikkumiseen saatiin käyttämällä vauhdin hidastamista hahmon lähestyessä kohdetta. Hahmon ja kohteen välisen etäisyyden pienentyessä myös hahmon nopeus pienentyy. Vaihtelevuutta eri hahmotyyppien väliseen liikkumiseen tuotiin lisäämällä vaihtoehtoja hidastamisen nopeuteen. Lisäksi käytettiin kerrointa, joka mahdollisti hidastamisvauhdin hienosäädön eri tilanteisiin.

8.3 Esteiden ohitus

Esteiden ohitus on käyttäytymismalli, joka ohjaa hahmoa väistämään eteen tulevia esteitä. Corrupted Culturassa esteet ovat erilaisia objekteja, kuten huonekaluja, joita pelaaja voi sijoitella pelimaailmaan haluamallansa tavalla. Esteiden väistäminen toteutettiin siten, että hahmo pystyy kiertämään etenemisreitilleen osuvan esteen kuvitteellista ympyrän kehää pitkin. Toisin sanoen hahmon etäisyys esteen keskipisteestä on koko ajan vähintään puolet esteen leveydestä. Varsinainen hahmon ohjaus pelin aikana tapahtuu eräänlaisen puskurialueen perusteella. Hahmon kulkusuuntaan asetetun puskurin leveys on hahmolle asetettu halkaisija ja pituus suhteutettuna hahmon nopeuteen. Mitä nopeammin hahmo

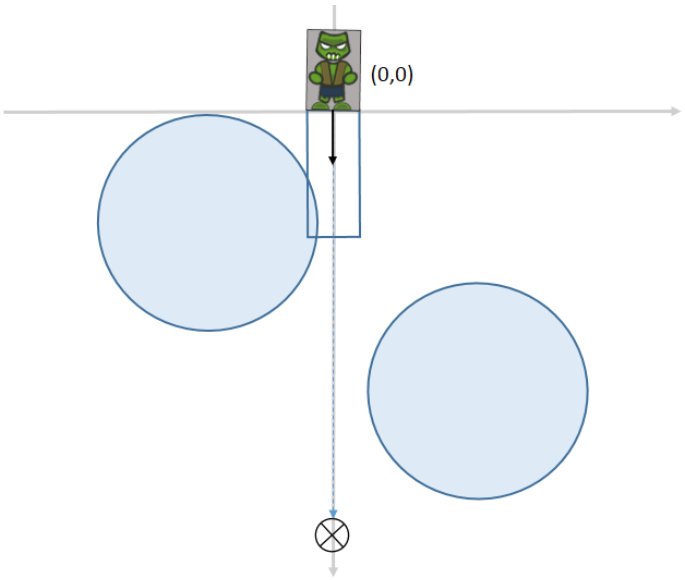
liikkuu, sitä pidempi puskurialue on. Puskurialueen sisällä havaittu este kääntää hahmon etenemissuuntaa siten, että se sivuuttaa kyseisen esteen. Puskurialueen kehittämisen tärkeimpänä etuna oli, että sillä pystyttiin ennakoimaan tulevat törmäykset ja tekemään tarvittava ohjaus niiden väistämiseksi riittävän ajoissa. Esteen ohitustilanne ja puskurialue on havainnollistettu kuvassa 14.

Idea esteiden ohittamiseen ympyrän kehää pitkin ja edellä kuvattua puskurialuetta käyttäen on lähtöisin Millingtonin ja Fungen ”Artifical Intelligence for Games” -teoksesta. Suunnittelu laskutoimituksineen sekä käytännön toteutus ovat kuitenkin täysin tämän opinnäytetyön tulosta.



Kuva 14. Vihollishahmo ja esteet pelimaailman koordinaatistossa

Puskurin toiminnan suunnittelu edellytti hahmon paikallisen koordinaatiston hyödyntämistä (kuva 15). Paikallisessa koordinaatistossa hahmon paikka sijaitsee origossa ja kulkusuunta sekä puskuri ovat sijoitettuina hahmon edessä olevalle akselille. Tämä menettelytapa helpotti törmäyksen aiheuttavien esteiden havainnollistamista ja tarvittavien laskutoimituksien tekemistä.

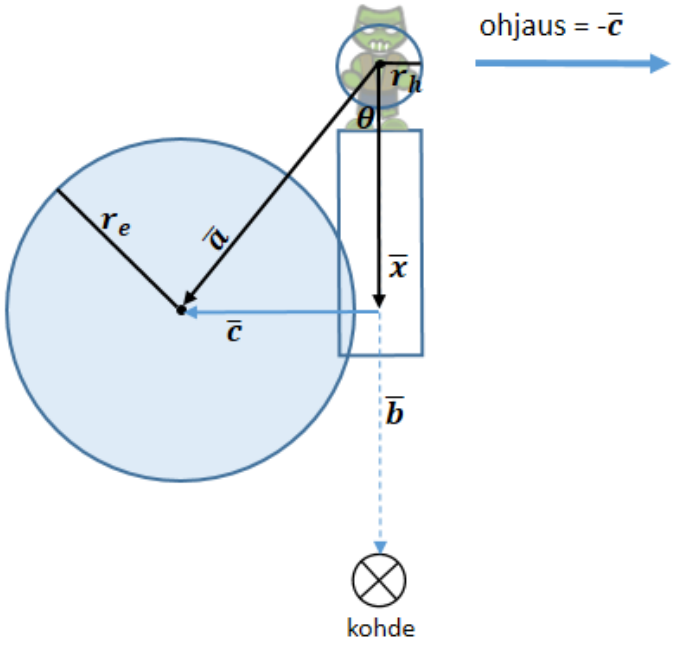


Kuva 15. Hahmon paikallinen koordinaatisto

Esteiden ohitus -käyttäytymismallin ensimmäisenä vaiheena on selvittää törmäyksen aiheuttavat esteet. Kuten kuvasta 15 voidaan huomata, kaikki hahmon taakse jäävät esteet ovat pois laskuista. Eli, jos hahmon y-koordinaatti on suurempi kuin jonkin esteen y-koordinaatti, ei tätä estettä tarvitse huomioida. Kun taakse jääneet esteet on karsittu pois, täytyy tarkastella, leikkaako joku jäljelle jääneistä puskurialueen.

8.3.1 Esteen poikkeama hahmon edessä olevalta akselilta

Puskurialueella mahdollisesti olevien esteiden selvittäminen aloitetaan ratkaisemalla esteen keskipisteen kohtisuora etäisyys hahmon kulkusuunnassa olevasta akselista. Kuvassa 16 tätä etäisyyttä kuvaa vektorin $\bar{c}$ pituus. Vektorin $\bar{c}$ laskemiseksi on tunnettava vektorit $\bar{a}$ ja $\bar{x}$. Vektori $\bar{a}$ on hahmon keskipisteestä esteen keskipisteeseen. Vektori $\bar{x}$ puolestaan kulkee hahmon kulkusuunnassa olevaa akselia pitkin hahmon keskipisteestä esteen keskipisteen tasolle.



$\bar{a}$ = vektori hahmon keskipisteestä esteen keskipisteeseen

$\bar{b}$ = vektori hahmon kulkusuuntaan

$\bar{c}$ = vektori hahmon edessä olevalta akselilta esteen keskipisteeseen

$\bar{x}$ = vektori hahmon keskipisteestä esteen keskipisteen tasolle

θ = vektoreiden $\bar{a}$ ja $\bar{b}$ välinen kulma

r_h= hahmon säde

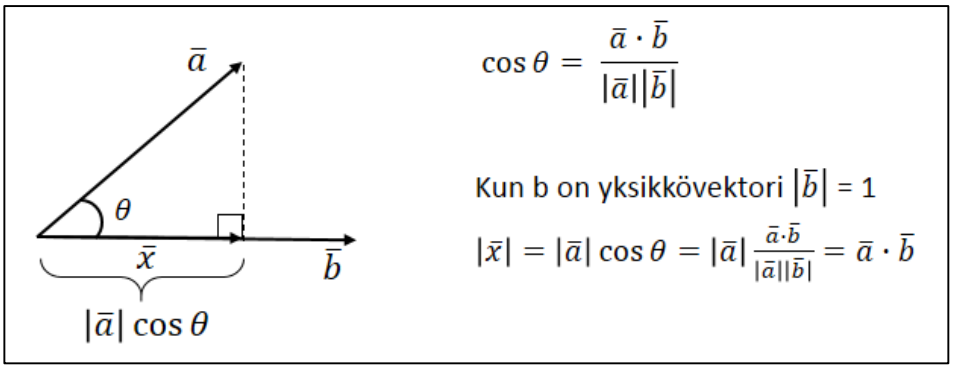
r_e= esteen säde

ohjaus = vektori, jonka mukaan hahmoa täytyy ohjata esteen väistämiseksi

Kuva 16. Esteen ohittamiseen käytettävän ohjausvektorin ratkaiseminen

Vektorin $\bar{x}$ pituus saadaan vektoreiden $\bar{a}$ ja $\bar{b}$ pistetulona, kuten kuvassa 17 on osoitettu. Vektori $\bar{b}$ on hahmon kulkusuunnan vektori ja kuvan 17 laskutoimitukset on johdettu siten, että vektorista $\bar{b}$ on käytetty sen yksikkövektoria. Yksikkövektorin pituus on 1, jolloin sillä ei ole vaikutusta kaavan tuottamaan tulokseen. Merkityksellistä on vain kyseisen vektorin suunta.

Kun vektorille $\bar{x}$ on saatu pituus, voidaan se ratkaista kokonaan kertomalla vektorin $\bar{b}$ yksikkövektorilla. Tällöin nämä vektorit ovat samansuuntaiset. Kuvan 16 vektori $\bar{c}$ eli esteen poikkeama kulkusuunnan akselilta saadaan vähentämällä vektorista $\bar{a}$ vektori $\bar{x}$.



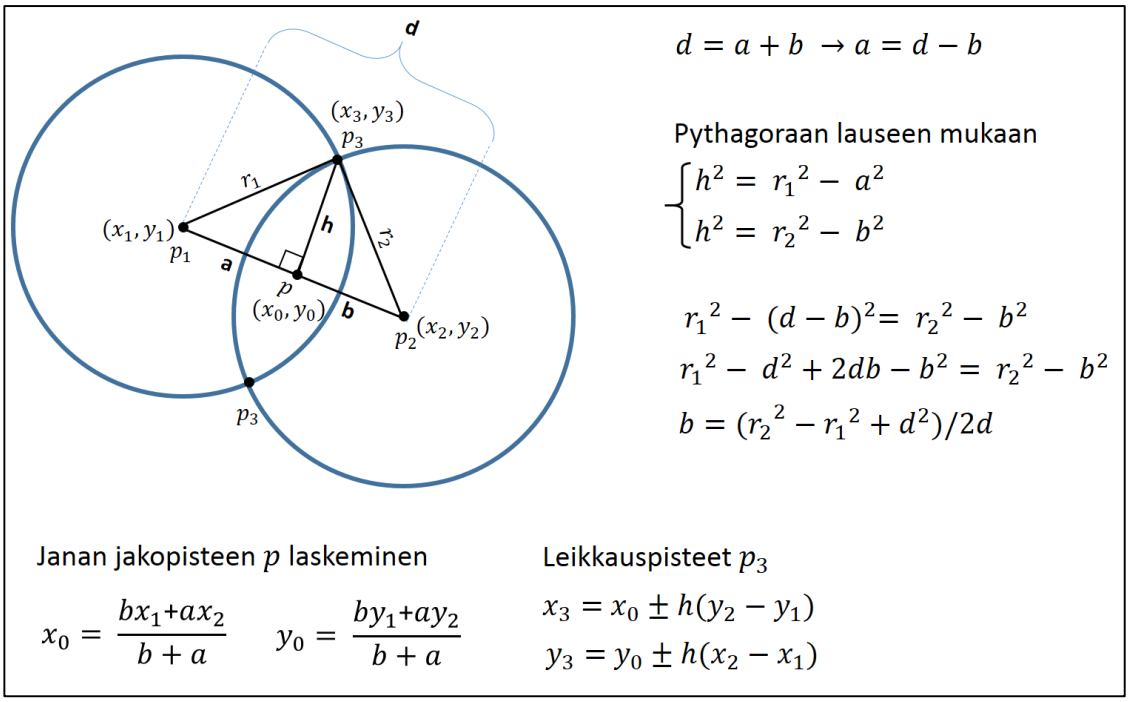
Kuva 17. Pistetulon käyttö ohjausvektorin ratkaisemisessa

8.3.2 Puskurialueen esteen tunnistaminen ja hahmon ohjaus

Aiemmin todettiin, että hahmon taakse jääviä esteitä ei tarvitse ottaa huomioon selvitettäessä törmäyksen aiheuttavia esteitä. Myöskin esteet puskurialueen etupuolella voi jättää huomiotta hahmon sen hetkisessä sijainnissa. Jäljelle jäävistä esteistä täytyy selvittää, leikkaavatko ne puskurialueen. Tämä onnistuu vertailemalla edellä ratkaistun vektorin $\bar{c}$ pituutta esteen säteen (r_e) ja hahmolle asetetun säteen (r_h) summaan. Mikäli kyseisen vektorin pituus on pienempi kuin säteiden summa, este on puskurialueen sisäpuolella (kuva 16). Tällöin esteen ohittamiseksi hahmoa täytyy ohjata vektorin $\bar{c}$ vastavektorilla.

8.3.3 Jumiutumisen estäminen

Käytettäessä edellä kuvattua ympyrän kehää pitkin tapahtuvaa esteen ohitusta havaittiin, että se ei toimi sellaisenaan kaikissa käyttötarkoituksissa. Ongelmia ilmenee, jos pelialueella on vähintään kaksi estettä sijoitettuna siten, että niiden kehät leikkaavat toisensa. Vihollishahmo yrittää kulkiessaan väistää molempia esteitä, jolloin se lopulta ajautuu leikkauspisteeseen eikä voi jatkaa matkaa. Ratkaisu ongelmaan oli leikkauspisteen selvittäminen ja uuden suunnan antaminen hahmolle, kun se on riittävän lähellä tätä pistettä.



Kuva 18. Ympyröiden leikkauspisteiden laskeminen

Ympyröiden leikkauspisteen selvittäminen ohjelmakoodissa tapahtuu kuvassa 18 esitetyillä laskutoimituksilla. Ensiksi täytyy selvittää Pythagoraan lausetta ja yhtälöparia käyttäen muuttujat a ja b. Tämän jälkeen näiden muuttujien avulla pystytään laskemaan ympyröiden keskipisteiden välisen janan jakopisteen p koordinaatit. Kyseisiä koordinaatteja käyttäen saadaan lopulta ratkaistua leikkauspisteiden koordinaatit.

8.4 A*-polunhakualgoritmi

Pelien tekoälyissä pisteestä pisteeseen kulkeminen on usein toteutettu käyttäen A* (A-tähti) nimistä algoritmia. Syynä sen suosioon on yksinkertainen toteutus, tehokkuus ja muokattavuus. Algoritmin avulla pystytään selvittämään edullisin polku, eli joukko pisteitä joiden kautta täytyy kulkea päästäkseen alkupisteestä loppupisteeseen. Siihen, mikä polku on ”edullisin”, pystytään vaikuttamaan erilaisilla heuristiikoilla. (Millington & Funge 2009, 215-216.)

Perehtyminen A*-algoritmiin aloitettiin opiskelemalla Millingtonin ja Fungen ”Artifical Intelligence for Games” -teosta, jossa oli esitetty erilaisten polunetsintään tarkoitettujen algoritmien toimintaa. Myös Corrupted Culturan tekoälyn suunnitte-

lussa ja toteutuksessa käytettiin apuna kyseisestä teoksesta löytyvää A*-algoritmin runkoa. Seuraavaksi selostetaan A*-algoritmin toimintaperiaate ja kuinka sitä on käytetty Corrupted Culturan vihollishahmojen tekoälyssä. Algoritmin toimintaan liittyvät luokat ja niiden ohjelmakoodi on esitetty liitteen 2 luvussa 5.

8.4.1 Pelialueen jakaminen ruudukoksi

A*-algoritmin käyttäminen edellyttää hahmon käyttämän pelialueen jakamista osiin (Tiles), kuten kuvassa 19 on esitetty. Pelikenttä jaettiin ruudukoksi (TileMap) käyttäen neliöitä, joista jokainen tallennettiin int-tyyppiseen kaksiulotteiseen taulukkoon. Näin jokainen taulukon alkio vastaa yhtä pelikentän osaa ja sen paikka taulukossa on sama kuin paikka pelikentän koordinaatistossa. Jokainen alkio saa arvoksi joko 1 tai 0 seuraavasti: 1 silloin, kun alkiota vastaava pelikentän osa sisältää väistettävän esteen, muuten 0. Kyseisen menettelyn avulla saadaan kerättyä tieto siitä, missä hahmo pystyy liikkumaan. Ruudukon muodostamisessa olisi voitu neliöiden sijaan käyttää myös suorakulmioita, kuusikulmioita, kolmioita tai mitä tahansa kuvioita. Neliöruudukko oli helpoiten omaksuttavissa ja se helpotti myös algoritmissa käytettävää laskentaa.

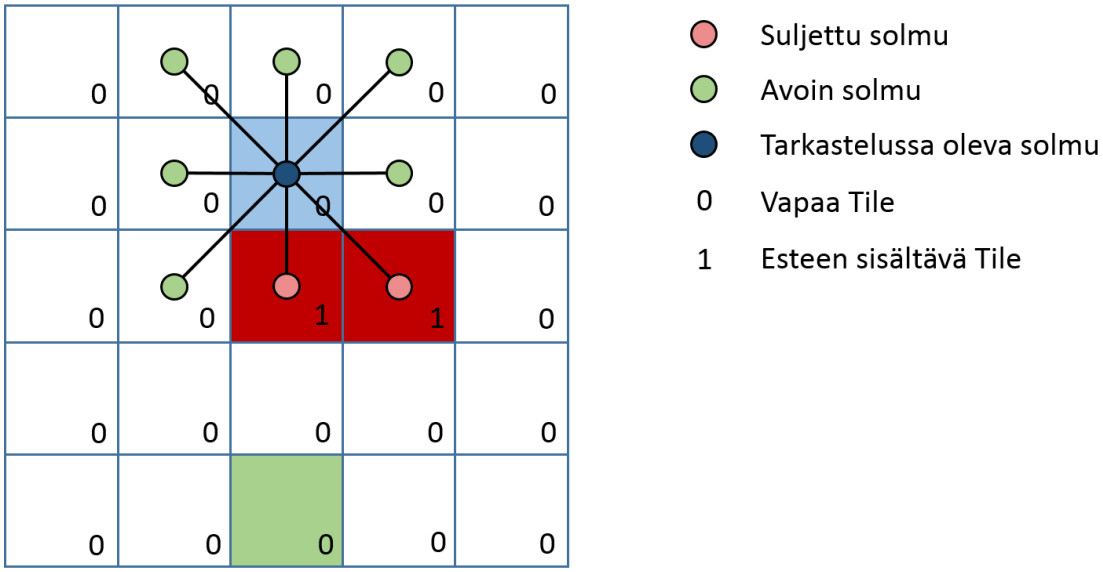
Pelialueen jakaminen ruudukoksi toteutettiin hahmokohtaisesti: mitä suurempi hahmo, sitä suurempi Tile. Hahmo käyttää yhden Tilen kokoisen alueen, jolloin hahmon koon kasvaessa Tilejen lukumäärä vähenee. Hahmokohtaista ruudukkoa käyttämällä pystyttiin vaikuttamaan siihen miten hahmo pelialueella liikkuu. Tällöin esimerkiksi pienikokoiset hahmot pystyvät kulkemaan väleistä, joista isot eivät mahdu.

8.4.2 Solmujen käyttö

Kun tieto pelialueesta on saatu taulukoitua, täytyy aloittaa solmujen, eli nodejen, ja niiden välisten yhteyksien tarkastelu. Jokaisen ruudukon neliön keskipisteeseen voi ajatella solmun, joka sitoo sen ympärillä olevien neliöiden solmuihin. Näin muodostuu solmujen verkosto, johon A*-algoritmi etsii edullisimman polun.

Solmujen tarkastelu alkaa polun ensimmäisestä solmusta, joka on sama kuin vihollishahmon alkusijainti. Tästä solmusta siirrytään kauemmaksi ympäröiviä solmuja tutkien, kunnes kohde löytyy. Tutkittaessa ympäröiviä solmuja ne lajitellaan

avointen ja suljettujen solmujen listaan. Avointen solmujen lista sisältää ne solmut, jotka saattavat kuulua lopulliseen polkuun. Tähän listaan kuuluvat kaikki muut solmut paitsi ne, joiden Tile sisältää esteen (pelikentän sisällön kaksiulotteisessa taulukossa Tileä vastaavan alkion arvo on 1). Suljettujen solmujen listaan puolestaan lisätään solmut, jotka on jo tarkastettu.



Kuva 19. Ruudukko (TileMap) ja solmut (nodet)

Kuvassa 19 on havainnollistettu tarkastelussa oleva solmu ja sitä ympäröivät solmut käyttäen 8-suuntaista tarkastelua (diagonaalinen etsintä). Vaihtoehtoisesti tarkastelun voisi suorittaa esimerkiksi vain neljään suuntaan: ylös, alas, vasemmalle ja oikealle. Tällöin lopullisen polun laskeminen vie vähemmän aikaa, mutta lopullisesta polusta ei tule kaikkein lyhyin ja luonnollisin. Suorituskykytestauksen perusteella 4-suuntaista tarkastelua käytettäessä prosessorin käyttö oli huomattavasti pienempi verrattuna 8-suuntaiseen.

Ympärillä olevien solmujen selvittämiseen käytettiin kuvassa 20 esitettyä ohjelmakoodia. Siinä muuttujat x ja y ovat tarkastelussa olevan solmun koordinaatit. Lopputuloksena on lista kaikkien kuvan 19 solmuverkoston solmujen sijainneista.

```
//Listataan kaikki mahdolliset ympärillä olevat nodet...
//Käytetään 8-suuntaista tarkastelua
List<Point> nodeSijainnit = new List<Point>();
nodeSijainnit.Add(new Point(x, y - 1));//Ylä
nodeSijainnit.Add(new Point(x, y + 1));//Ala
nodeSijainnit.Add(new Point(x - 1, y));//Vasen
nodeSijainnit.Add(new Point(x + 1, y));//Oikea
nodeSijainnit.Add(new Point(x - 1, y - 1));//Ylävasen
nodeSijainnit.Add(new Point(x - 1, y + 1));//Alavasen
nodeSijainnit.Add(new Point(x + 1, y - 1));//Yläoikea
nodeSijainnit.Add(new Point(x + 1, y + 1));//Alaoikea
```

Kuva 20. Ympäröivät solmut ohjelmakoodissa

8.4.3 Seuraavan solmun valinta

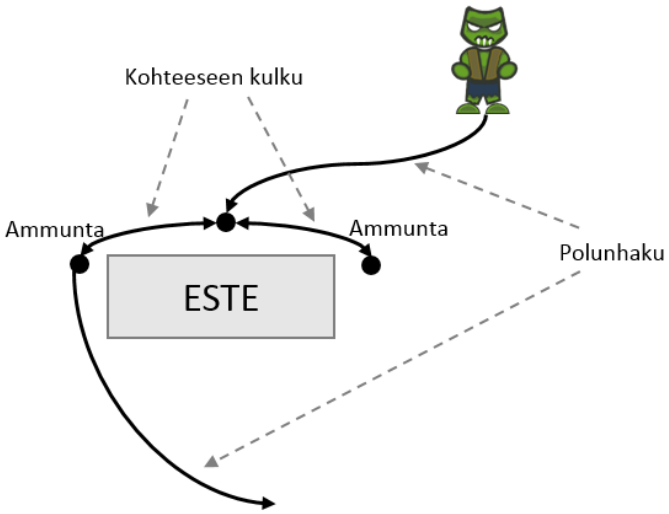
Kaikki ympäröivät vapaat solmut lisätään avointen solmujen listaan, lukuun ottamatta niitä, jotka ovat jo avointen tai suljettujen listalla. Jokaiselle lisättävälle solmulle lasketaan niin sanottu F-arvo, joka on arvio solmun kautta kulkevan lopullisen polun pituudelle. Tämä saadaan, kun lasketaan yhteen aloitussolmusta kuljettu matka (G) ja arvioitu matka päätössolmuun (H). Ympäröivistä solmuista pienimmän F-arvon omaava solmu valitaan lopullisen polun seuraavaksi solmuksi. Jokaiselle valitulle solmulle asetetaan myös isäntäsolmu, joka on sen hetkinen tarkastelussa oleva solmu. Tämän jälkeen siirrytään tarkastelemaan valittua solmua ja toistetaan edellä kuvatut toimenpiteet. Tätä jatketaan, kunnes tarkastelussa oleva solmu on itse päätössolmu, mikä tarkoittaa kohteen löytymistä.

H-arvo on pelkkä arvio matkasta päätössolmuun, koska todellista matkaa ei tiedetä ennen kuin lopullinen polku on selvitetty. Todellinen matka päätössolmuun on arvioitua pidempi, jos vastaan tulee esteitä. Arvio H-arvosta lasketaan käyttäen jotain heuristista funktiota. Heuristiikan valinta vaikuttaa algoritmin tehokkuuteen ja siihen, onko löydetty reitti kaikkein lyhyin ja edullisin. Funktiota muuttamalla algoritmin saa tuottamaan erilaisia polkuja. Corrupted Culturan hahmojen tekoälyn heuristisena funktiona käytettiin tarkastelussa olevan solmun ja päätössolmun välistä suoraa etäisyyttä, joka laskettiin kahden pisteen välisen etäisyyden kaavalla (1).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

8.5 Piiloutuminen

Vaatimuksena oli, että pelin hahmot osaavat käyttää pelialueen esteitä myös piiloutumiseen. Piiloutuminen toteutettiin edellä kuvattuja tekniikoita hyödyntäen: Hahmo ohjautuu esteen taakse polunhakualgoritmin avulla, minkä jälkeen se siirtyy satunnaisesti esteen puolelta toiselle käyttäen kohteeseen kulkua. Ollessaan esteen sivulla hahmo suorittaa ampumistoiminnon ja palaa takaisin piiloon. Hahmo poistuu piilosta laskemalla polunhakua käyttäen uuden reitin kohteeseen. Piiloutumisessa käytettävät tekniikat on esitetty kuvassa 21.

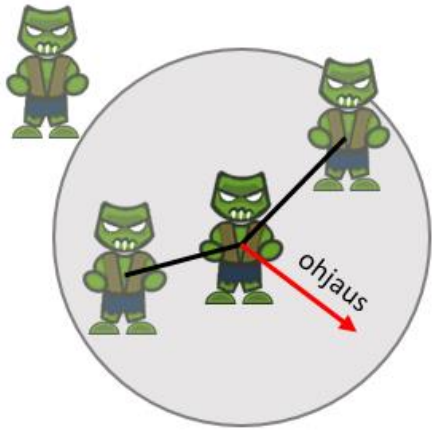


Kuva 21. Piiloutumisen toteutus

8.6 Ympäröivien hahmojen tunnistus

Jos pelaajalle halutaan luoda illuusio älykkäästä toiminnasta, pelkästään pelimaailmassa olevien esteiden huomioiminen ei riitä, vaan hahmon pitää liikkuessaan pystyä reagoimaan myös ympärillä oleviin toisiin hahmoihin. Pelin pelattavuuden ja viihdyttävyyden kannalta on välttämätöntä, että pelissä esiintyvät hahmot pystyvät käyttäytymään myös ryhmänä. Corrupted Culturan hahmojen tekoälyltä vaadittiin, että hahmot eivät törmäile tai jumiudu toisiinsa eivätkä kulje toistensa lävitse. Vaatimukset täytettiin loitontamalla hahmoa muista tietyllä etäisyydellä olevista hahmoista.

Hahmon loitontamisesta vastaa Loitonnus-luokka (liite 2, luku 7), josta löytyy metodit ympäröivien hahmojen selvittämiseen sekä varsinaisen loitonnuksen laskemiseen. Loitonnusta laskiessa selvitetään vektorit hahmosta jokaiseen ympäröivään hahmoon. Jokaista lähellä olevaa hahmoa kohti on ohjausvektori, joka on kääntäen verrannollinen hahmojen väliseen etäisyyteen nähden – mitä lähempänä hahmot toisiaan ovat, sitä suurempi ohjausvektori on. Ohjausvektoreiden summana saadaan lopulta tarvittava ohjaus hahmon loitontamiseksi muista hahmoista. Tilannetta havainnollistaa kuva 22.



Kuva 22. Hahmon loitonnus ympäröivistä hahmoista

8.7 Päällekkäisyyden hallinta

Vaikka jokainen hahmo tunnistaa ympärillään olevat muut hahmot ja osaa aiemmin kuvattua loitonnus-tekniikkaa käyttäen siirtyä kauemmas niistä, voi silti syntyä tilanne, jossa kaksi hahmoa ajautuvat päällekkäin. Tällaisen tilanteen välttämiseksi täytyy pelilogiikan päivityksen yhteydessä tarkastaa kaikki hahmojen väliset etäisyydet. Mikäli kahden hahmon välinen etäisyys toisistaan on pienempi kuin niille asetettujen säteiden summa (kuva 23), siirretään hahmoja päällekkäisyyden verran kauemmas toisistaan.

Kuva 23. Hahmojen päällekkäisyys

Edellä kuvattua päällekkäisyyden hallintaa jouduttiin käyttämään myös ympyrän kehää pitkin tapahtuvassa esteen ohituksessa. Lisäksi hahmon kulkiessa polunhakualgoritmilla laskettua reittiä, täytyi huolehtia, ettei hahmo joudu päällekkäin esteelle asetetun pohjan kanssa. Päällekkäisyyden hallinnan hoitava ohjelmakoodi on esitetty liitteen 2 luvussa 8.

8.8 Päätöksenteko ja tilanhallinta

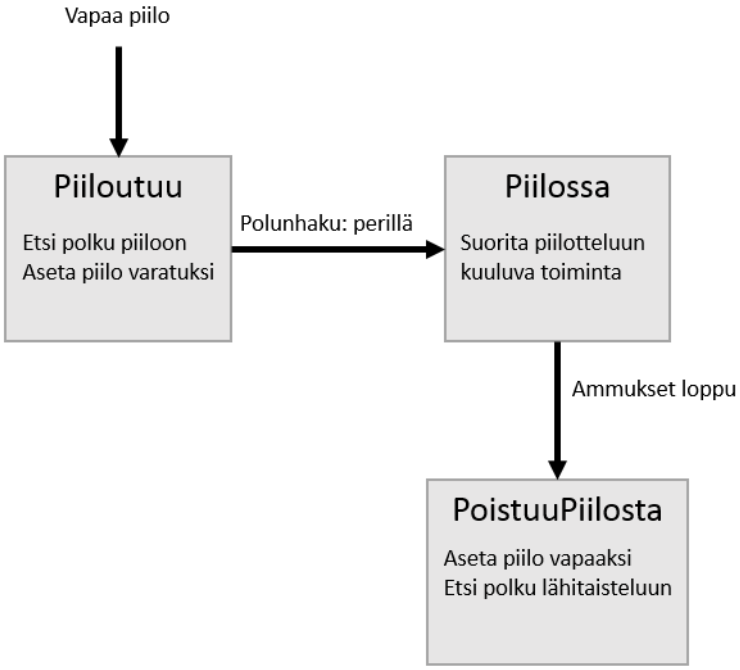
Kuten jo luvussa 5.5.2 mainittiin, päätöksenteolla on merkittävä osa pelien tekoälyissä. Päätöksenteko Corrupted Culturassa hoidettiin käyttäen yksinkertaista tilakonetta (Tilanhallinta-luokka), joka koostuu 12 erilaisesta tilasta (kuva 24).

```
public enum Tilat
{
    Sisaantulo,
    AutoPilot,
    Piiloutuu,
    Piilossa,
    PoistuuPiilosta,
    Ansassa,
    Lahitaistelussa,
    Jumissa,
    PolunHaku,
    KiertaaAnsaa,
    OttaaOsumaa,
    Kuolee
}
```

Kuva 24. Tilakoneen tilat

Tila määrittää sen, mitä Tekoaly-luokan käyttäytymismallia milloinkin suoritetaan. Käyttäytymismalleja käytetään myös siirryttäessä tilasta toiseen, minkä pelaaja

havaitsee hahmon päätöksentekona. Tilakoneen käyttö on havainnollistettu kuvassa 25. Siinä on mallinnettu tilakaaviota käyttäen hahmojen piiloutumiseen liittyvät kolme tilaa: ”Piiloutuu”, ”Piilossa” ja ”PoistuuPiilosta”. Mikäli hahmo havaitsee vapaana olevan piilon, se tekee päätöksen piiloutumisesta ja sen tilaksi tulee ”Piiloutuu”. Tällöin hahmo etsii polun kyseiseen piiloon ja lähtee kulkemaan sitä pitkin. Kun hahmo on saapunut perille, vaihtuu sen tilaksi ”Piilossa”, jolloin se suorittaa piilottelulle ominaista toimintaa. Kun hahmon ammukset loppuvat, se tekee päätöksen piilosta poistumisesta. Samalla sen tilaksi tulee ”PoistuuPiilosta”, jolloin se etsii polun lähitaisteluetäisyydelle ja lähtee kulkemaan tätä polkua pitkin.



Kuva 25. Piiloutumisen tilakaavio

Tilakonetta ja valittua tilaa käytettiin myös animaatioiden toteutuksessa. Hahmon tila määrää sen, mitä animaatiota milloinkin käytetään.

8.9 Piirtojärjestyksen hallinta

Jotta näytöllä esitettävä sisältö olisi luonnollinen, täytyy piirrettävillä kohteilla olla piirtojärjestys. Jos esimerkiksi vihollishahmo kulkee esteen etupuolelta, täytyy hahmo piirtää esteen päälle. Jos vihollishahmo taas on esteen takana piilossa,

täytyy este olla piirrettynä päällimmäisenä. Piirtojärjestyksen hallitsemiseksi luotiin Piirtojärjestys-luokka ja IPiirtojärjestys-rajapintaluokka, jonka kaikki piirrettävät kohteet toteuttavat. Rajapintaluokassa on yksi int-tyyppinen ominaisuus ”alareunaY”, joka on mahdollista vain palauttaa. Jokainen rajapinnan toteuttava luokka käyttää tätä ominaisuutta ja palauttaa alareunansa y-koordinaatin.

Rajapinnan toteuttavien luokkien instanssit voidaan lisätä List-kokoelmaan, jonka alkioiden tyypiksi on määritetty IPiirtojärjestys (kuva 26). Tähän kerätään kaikki ne oliot, joiden piirtojärjestystä halutaan hallita. Olio voidaan lisätä kokoelmaan vain, jos se on toteuttanut IPiirtojärjestys-rajapinnan.

```
public List<IPiirtojarjestys> piirrettavat;
```

Kuva 26. List-kokoelma IPiirtojarjestys-rajapinnan toteuttaville olioille

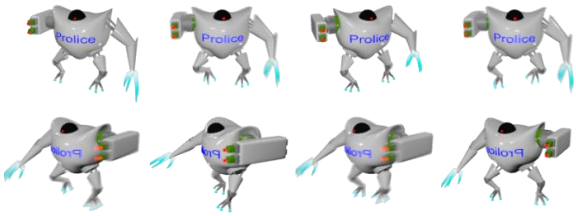
Piirtojärjestyksen hallinta tapahtuu Piirtojärjestys-luokassa, jolle välitetään viittaus edellä kuvattuun kokoelmaan. Luokka sisältää Jarjesta()-metodin, joka lajittelee kokoelman alkiot y-koordinaattien mukaan pienimmästä suurimpaan. Tämän jälkeen jokaiselle alkiolle asetetaan DrawOrder-arvoksi sen paikka kokoelmassa. DrawOrder-ominaisuus on käytössä kaikilla DrawableGameComponent-luokasta periytyvien luokkien instansseilla. Kun arvo on asetettu, XNA:n pelisilmukka osaa automaattisesti piirtää kohteet oikeassa järjestyksessä.

8.10 Animaatiot

Vaikka animaatioita ei lasketa kuuluvaksi tekoälyyn, on nekin huomioitava suunniteltaessa ja toteutettaessa hahmojen liikkeitä. Corrupted Culturan tekoälyn toteutuksen yhteydessä toteutettiin myös vihollishahmojen liike- ja tuhoutumisanimaatiot.

Animaatiot luotiin käyttäen sprite sheet -kuvia (kuva 27). Sprite sheet on kuvasarja, josta luodaan animaatio näyttäen yksittäisiä kuvia (frame) vuorotellen. Kuvasta toiseen siirtymiseen käytettiin ajastinta: kun kuvan näyttöaika tulee täyteen, siirrytään näyttämään seuraavaa kuvaa. Mikäli näytettävä kuva on kuvasarjan viimeinen, seuraava kuva on kuvasarjan ensimmäinen. Ajastimen aika päivittyy

pelilogiikan päivityksen yhteydessä. Näytettävän kuvan piirto tapahtuu välittämällä tieto sen sijainnista kuvasarjassa spriteBatch-olion Draw-metodille. Näin kyseinen metodi osaa esittää näytöllä juuri oikean kuvasarjan kuvan.

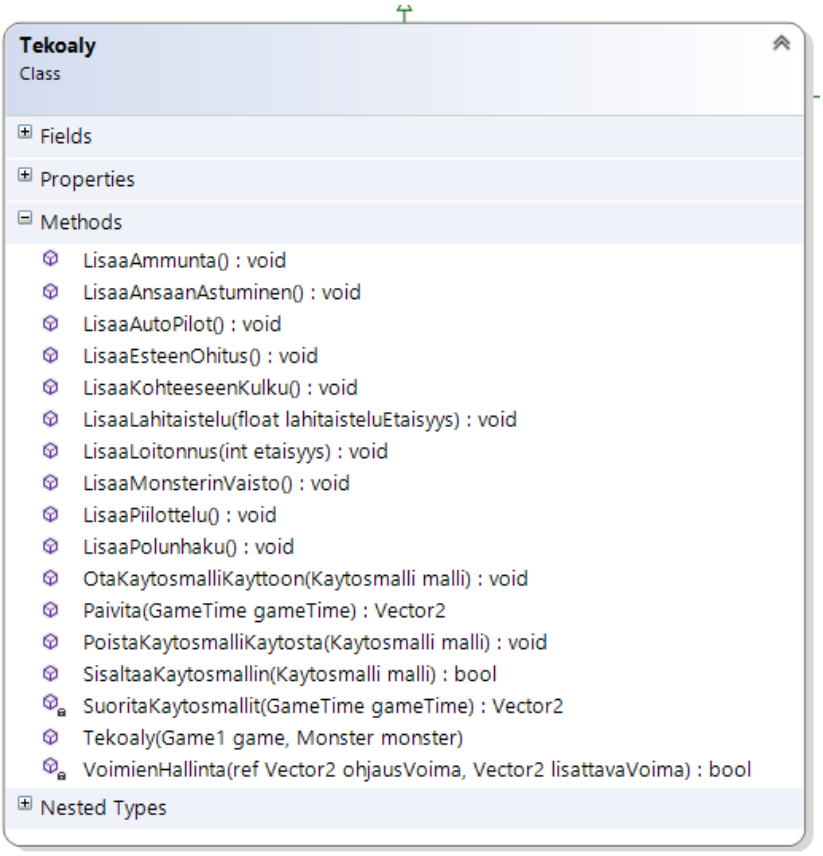


Kuva 27. Sprite sheet (Jontka 2013)

8.11 Tekoälyn asettaminen hahmolle

Asiakas käyttää lopullista tekoälyä luomalla pelissään Monster-luokan instansseja tietyillä parametreilla. Parametrit määräävät hahmon alkusijainnin ja tyypin, jonka mukaan hahmolle asetetaan tyyppiä vastaava tekoäly. Monster-oliolla on ominaisuutena Tekoaly-luokan olio, ja varsinainen tekoälyn asettaminen tapahtuu tämän olion metodikutsuina Monster-olion muodostimessa. Näin eri hahmotyyppien käyttämiä käyttäytymismalleja pystytään helposti muuttamaan. Tekoaly-luokan metodit on esitetty kuvassa 28.

Varsinainen tekoäly pyrittiin rakentamaan mahdollisimman modulaarisesti, jolloin eri käyttäytymismallien hallinta Tekoaly-luokan avulla on mahdollista. Modulaarinen rakenne helpottaa myös mahdollista tekoälyn jatkokehitystä.



Kuva 28. Tekoaly-luokan metodit

9 Mobiililaitteen resurssit ja tekoälyn testaus

Tekoälyn kehittäjien suurimpana rajoitteena ovat laitteen asettamat fyysiset rajoitteet. Kehittäjät joutuvat usein käyttämään paljon aikaa siihen, että tekoälyn prosessointiaika ja muistinkäyttö saadaan vaaditulle tasolle. (Millington & Funge 2009, 25.)

Tämän opinnäytetyön tekoälyä ohjelmoidessa ja iteraatiovaiheiden testauksissa havaittiin heti alkuvaiheessa, että mobiililaitteen, tässä tapauksessa älypuhelimen, tarjoamat resurssit ovat erittäin rajalliset. Tämä asetti omat haasteensa ohjelmointiin. Tekoälyn vaatimaa muistin ja prosessorin käytön määrää oli valvottava ja ohjelmakoodi täytyi saada niiden osalta mahdollisimman kevyeksi. Keinot, joilla liian suurta muistin käyttöä rajoitettiin, olivat olioiden mahdollisimman pikainen tuhoaminen ja niiden varaaman muistin vapauttaminen sekä suurten taulukoiden ja listojen välttäminen. Prosessorin käyttöä puolestaan säästettiin karsimalla turhia silmukoita ja välttämällä useiden sisäkkäisten silmukoiden käyttöä.

Ohjelmakoodin optimointi etenkin A*-polunhakualgoritmia tehtäessä vaati jatkuvaa kuormitus- ja suorituskykytestausta.

Resurssien riittävyyden testaamiseksi piti tehdä myös toimivuustestausta, jolla varmistuttiin tehdyn työn toimivuudesta. Molemmat testaukset suoritettiin jokaiselle komponentille iteraation aikana sekä liitettäessä komponenttia osaksi tekoälyä. Seuraavaksi kerrotaan molempien testauksien suorittamisesta sekä esitetään niiden antamia tuloksia.

9.1 Suorituskykytestaus

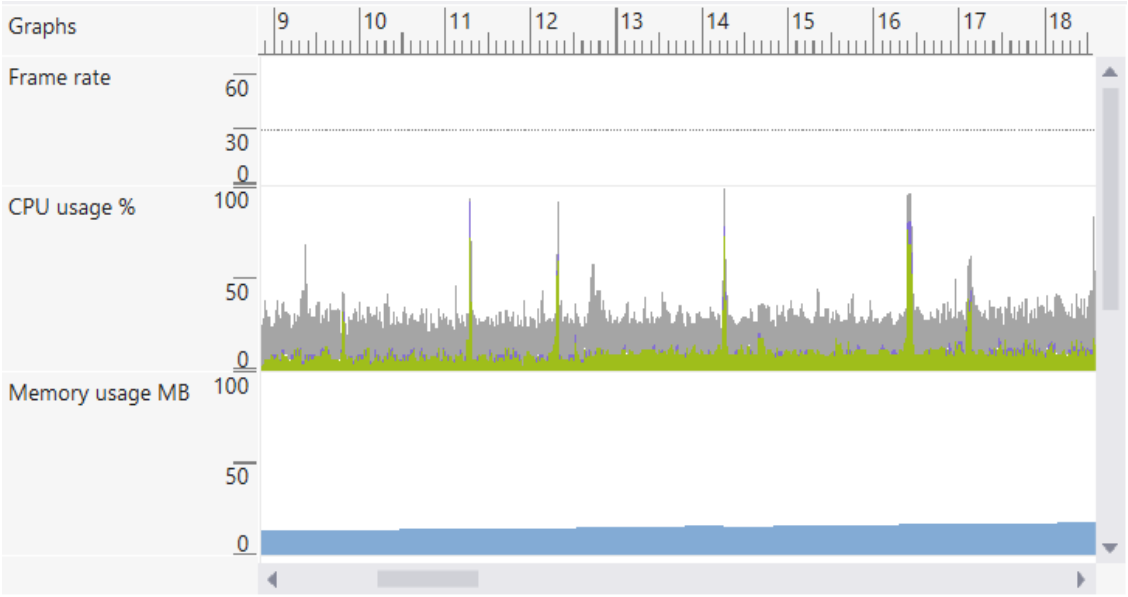
SDK 8 -kehitystyökaluihin kuuluu Windows Phone Application Analysis -työkalu, jolla pystyy testaamaan, kuinka paljon puhelimen resursseja sovellus käyttää. Työkalu on integroituna Visual Studioon ja sitä voi käyttää testattaessa sovellusta joko emulaattorilla tai fyysisellä laitteella. Tämän opinnäytetyön suorituskykytestaukset tehtiin ajamalla sovellusta Nokia Lumia 800 -älypuhelimessa, joka sisälsi 1.4 GHz:n prosessorin ja 512 megatavua RAM-muistia. Lisäksi tekoälyn suorituskykyä testattiin projektin lopussa myös Nokia Lumia 820 -laitteella, joka sisälsi 1.5 GHz:n tuplaydinprosessorin ja 1 gigatavua RAM-muistia. Tässä kappaleessa esitetyt suorituskykytestaukset ja niiden tulokset ovat peräisin Lumia 800 -laitteesta.

Windows Phone Application Analysis -työkalua käytettäessä täytyy ensiksi valita, mitä resursseja halutaan seurata. Tämän jälkeen työkalu käynnistää sovelluksen ja aloittaa datan keräämisen kyseisten resurssien käytöstä. Kun sovelluksen suorittaminen lopetetaan, työkalu kokoaa yhteenvedon seuratuista resursseista ja esittää ne graafisesti. Työkalulla pystyy seuraamaan lukuisia resursseja akun kulumisesta verkon käyttöön, mutta tekoälyn kehittämisen ja testaamisen kannalta oleellisia olivat vain käytetyn muistin ja prosessoritehon määrä.

Varsinainen testaus suoritettiin käyttämällä sovellusta ja sen ominaisuuksia, kuten loppukäyttäjä eli pelaaja niitä käyttäisi. Suorituskykyä testattiin myös kuormitustilanteissa, jotka järjestettiin lisäämällä tekoälyn laskentaa ja muistia tarvitsevien hahmojen määrää. Kuvat 29 - 31 ovat esimerkkejä erilaisten testitapausten tuloksista, jotka on saatu Windows Phone Application Analysis -työkalulla. Kaavioiden värit tarkoittavat seuraavaa:

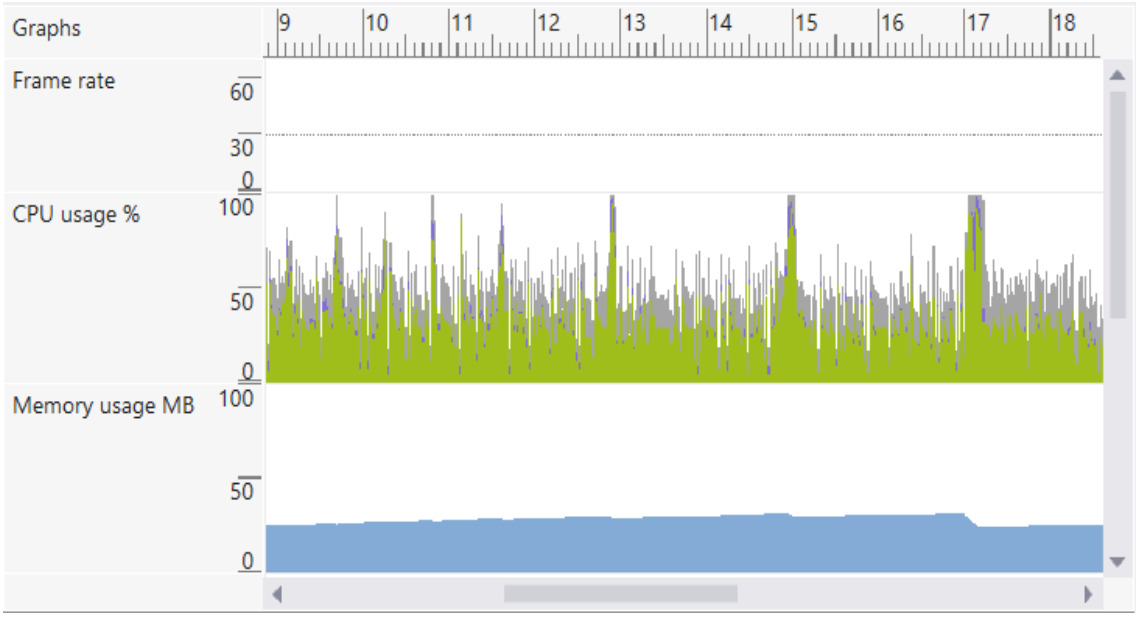
- Sininen kuvaa käytetyn muistin määrää megatavuissa.
- Vihreä ilmaisee ruudun päivitystä ja kosketussyötteitä.
- Harmaa tarkoittaa sovelluksen ulkopuolista toimintaa. Mikäli harmaa alue on suuri, jokin toinen taustalla oleva sovellus vaikuttaa testattavan sovelluksen suorituskykyyn.
- Violetti ilmaisee sovelluksen toimintaa, joka ei liity käyttöliittymään. Tällaisia voivat olla esimerkiksi taustasäikeet.
- Valkoinen kuvaa prosessorin ja muistin vapaina olevia osuuksia. Mitä enemmän kaavio sisältää valkoista, sitä paremmin sovellus toimii ja reagoi.

Kuva 29 esittää kahdeksan hahmon ja niiden tekoälyn käyttämiä resursseja. Kahdeksan on tyypillinen yhtaikaa esiintyvien vihollishahmojen lukumäärä Corrupted Culturassa. Kuten kuvasta havaitaan, sovelluksen keskimääräinen prosessorin käyttö (vihreä väri) on noin 20 prosenttia ja keskusmuistin varaus (sininen väri) noin 20 megatavua. Tällaisessa tilanteessa peli toimii sulavasti ja reagoi nopeasti pelaajan syötteisiin.



Kuva 29. Kahdeksan hahmon tekoälyn vaatimat resurssit

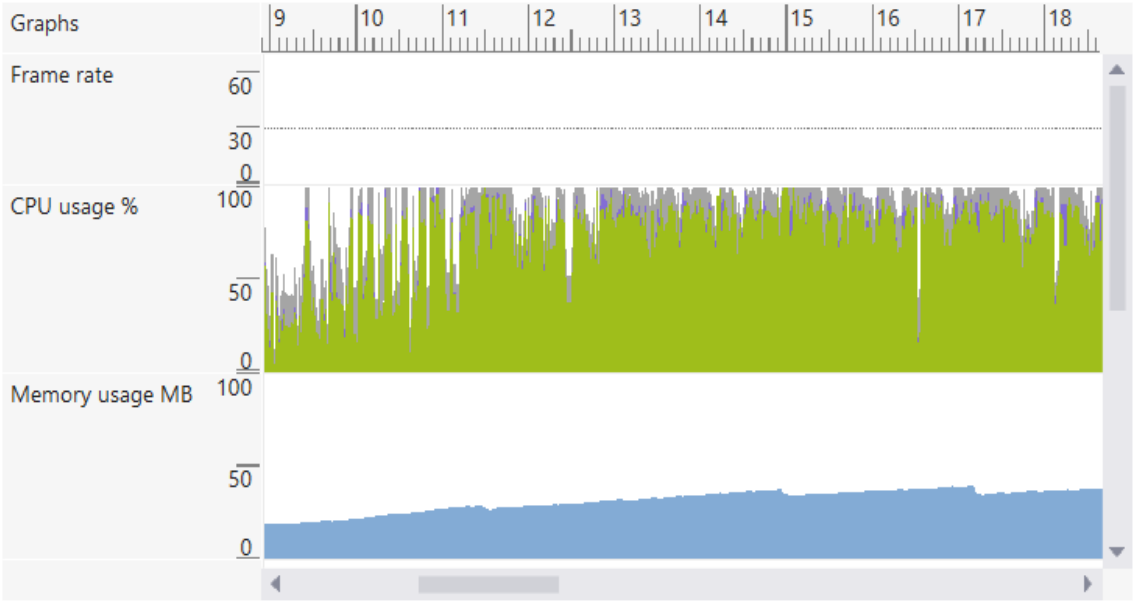
Kuvan 30 esittämässä testitapauksessa hahmojen lukumäärä nostettiin kahteenkymmeneen. Samalla keskimääräinen prosessoritehon käyttö nousi noin 50 prosenttiin ja tarvittava muistin määrä oli noin 30 megatavua. Käytetty Lumia 800 -älypuhelin selvisi ongelmitta myös tällaisesta kuormituksesta.



Kuva 30. Kahdenkymmenen hahmon tekoälyn vaatimat resurssit

Kuvan 31 esittämät tulokset havainnollistavat testitapauksen, jossa laitteen resurssit eivät riitä sovelluksen suorittamiseen. Kyseisessä testitapauksessa käytettiin kahtakymmentä vihollishahmoa, joiden tekoäly käytti jatkuvaa polunhakua. Jatkuvassa polunhaussa tekoäly suorittaa polunhaku-algoritmin, jokaisen Update-metodikutsun yhteydessä. Tämä nostaa prosessorin käytön lähes jatkuvasti 100 prosenttiin, mikä tarkoittaa, että koko ajan joudutaan tekemään tekoälyn vaatimaa laskentaa.

Edellä kuvatussa tilanteessa XNA:n pelisilmukka toimii siten, että se keskittyy vain pelilogiikan päivittämiseen eli Update-metodin suorittamiseen. Sen seurauksena Draw-metodi ja sisällön piirto jää kokonaan tekemättä, mikä näkyy pelaajalle hahmojen liikkeen nykimisenä. Jatkettaessa suorittamista tällaisella kuormituksella, sovellus kaatuu lopulta kokonaan.



Kuva 31. Kahdenkymmenen hahmon tekoälyn vaatimat resurssit käytettäessä jatkuvaa polunhakua

9.2 Toimivuustestaus

Toimivuustestauksessa testattiin tekoälyn toimintaa ja hahmojen liikkumista toimintavarmuuden näkökulmasta. Erilaisia testitapauksia luotin muun muassa muuttamalla:

- hahmojen lukumäärää ja tyyppiä
- hahmon käyttämää tekoälyä (lisäämällä/poistamalla/muokkaamalla tekoälyn komponentteja)
- testattavaa komponenttia
- hahmon alkusijaintia
- esteiden sijaintia
- ansojen sijaintia.

Varsinainen testin suorittaminen tapahtui käyttämällä sovellusta, kuten loppukäyttäjä eli pelaaja sitä käyttäisi. Kukin testitapaus kesti noin kaksi minuuttia ja sen aikana varmistettiin tekoälyn toiminta oletetulla tavalla. Yleisimmät syyt toimivuustestauksen hylättyyn lopputulokseen olivat hahmon jumiutuminen ja reagoimattomuus syötteeseen sekä hahmon ajautuminen esteen päälle tai sen läpi.

10 Yhteenveto ja pohdinta

Tämän opinnäytetyön tarkoituksena oli suunnitella jo toteuttaa tekoäly Jontka osk:n kehittämään Corrupted Cultura -mobiilipeliin. Corrupted Cultura on suunnattu Windows Phone -käyttöjärjestelmille, joten keskeisimmät tekniikat olivat C#-ohjelmointikieli ja XNA. Kaiken kaikkiaan sovelluskehitysprojekti onnistui ja täytti sille asetetut vaatimukset. Tuloksena saatiin vaatimusten mukainen tekoäly, jota voidaan käyttää eri tavoin erilaisilla vihollishahmoilla. Tekoäly hoitaa vihollishahmojen liikkumisen ja päätöksenteon. Toteutetun tekoälyn runkoa voidaan käyttää sellaisenaan missä tahansa XNA-ohjelmistokehystä käyttävässä 2D-pelissä. Lisäksi tekoälyn luokkia voidaan pienillä muutoksilla hyödyntää muissakin C#-kielen peliprojekteissa.

Tekoälyn lisäksi opinnäytetyötä tehdessä toteutettiin myös tekoälyä käyttävät vihollishahmot ja aloitettiin niiden animaatioiden tekemistä. Tässä mielessä toteutui luvussa 5.3 mainittu määritelmä: pelien tekoälyä ei pitäisi kutsua sanalla ”tekoäly”, vaan parempi nimi sille olisi ”hahmosuunnittelu”.

Vihollishahmojen grafiikan puutteen vuoksi tekoälyn toiminnan testaus ja tyyppikohtainen hienosäätö pystyttiin tekemään vain yhdellä vihollishahmotyypillä. Samasta syystä myöskään tekoälyn lähitaistelu-osuutta ja kaikkia animaatioita ei voitu toteuttaa loppuun asti. Edellä mainittujen asioiden tekemättä jääminen aiheutui täysin tästä projektista riippumattomista syistä.

Opinnäytetyön lopputulos on esitetty kuvassa 32. Siinä tekoälyn kehityksessä käytettyä pelin prototyyppiä ajetaan Nokia Lumia 820 laitteella. Kuvan hahmot liikkuvat ja toimivat pelimaailmassa tekoälyn määräämällä tavalla.

Projektin kulku ja haasteet

Opinnäytetyön aiheen valinta tapahtui tammikuussa 2013, mutta varsinainen työskentely päästiin aloittamaan helmikuun alussa. Tavoitteena oli saada asiakkaan käyttöön toimiva tekoäly huhtikuun alussa, joten projektin aikataulu oli todella tiukka. Projektin onnistumisen varmistamiseksi luotiin projektisuunnitelma, jonka tärkein tehtävä oli asettaa projektille aikataulu ja vaaditut tehtävät. Suunni-

telma muuttui hieman projektin edetessä, mutta pääkohdat pysyivät alkuperäisinä. Projektille asetettu aikataulu ja välitavoitteet toteutuivat, kuten oli suunniteltu, mikä oli merkittävää projektin onnistumisen kannalta.

Koska projekti oli laaja ja aikaa niukasti, katsottiin alussa parhaaksi lähteä kehittämään tekoälyä ketterän ohjelmistokehityksen periaatteiden mukaan. Aikaa ei käytetty tarkan määrittelydokumentaation luomiseen, vaan asiakkaan kanssa listattiin vain tärkeimmät tekoälyltä vaaditut ominaisuudet. Tiivis yhteistyö asiakkaan kanssa mahdollisti määrittelyn täydentymisen projektin edetessä. Sovelluskehitysprojektin läpivientiin ei käytetty suoraan mitään ketterän ohjelmistokehityksen mallia, vaan alussa luotiin oma toimintamalli ketteryyden periaatteita ja käytäntöjä hyödyntäen. Tilanteessa, jossa asioista ei ole ennestään kokemusta ja uusia asioita joutuu opiskelemaan jatkuvasti työn edetessä, kyseinen ratkaisu osoittautui toimivaksi.

Työskentelin asiakkaan tiloissa, ja kirjaimellisesti saman pöydän ääressä, noin kolmena päivänä viikossa koko projektin ajan. Tällä oli varmasti suuri vaikutus projektin onnistumiseen. Asiakas oli jatkuvasti tietoinen siitä, mitä oltiin tekemässä ja mitä tehtäisiin seuraavaksi. Näin tekoäly pystyi kehittymään asiakkaan vaatimusten ja toiveiden mukaan. Asiakkaan puolelta löytyi tarvittaessa myös asiantuntevaa apua ja ideoita ongelmien ratkaisemiseksi. Toisaalta myös minulta kysyttiin näkemyksiä yleisesti Corrupted Culturan kehittämiseen liittyvissä asioissa.

Haasteita tekoälyn suunnittelussa ja toteutuksessa tarjosi tiukan aikataulun lisäksi mobiililaitteiden rajalliset resurssit. Sovelluksen mahdollisimman pieni prosessoritehon käyttö täytyi pitää koko ajan mielessä ohjelmoitaessa ja ratkaisuja suunniteltaessa. Resurssit riittivät hyvin kinemaattisten ja dynaamisten liikkumisalgoritmien käyttöön, mutta kyseiset algoritmit yksin osoittautuivat riittämättömäksi tekoälyn toteuttamiseksi. Näiden lisäksi täytyi käyttää A*-algoritmiin pohjautuvaa polunhakualgoritmia, jonka vaatima laskentateho ja muistin määrä olivat huomattavasti suuremmat. Rajalliset resurssit vaativat jatkuvan suorituskykytestauksen tekoa.

Haasteensa tarjosi myös erilaisten matemaattisten ongelmien ratkaiseminen. Itselleni tuli yllätyksenä, kuinka paljon matematiikkaa ja etenkin vektoreita peliohjelmointiin liittyy.

Jatkokehitys

Kuten jo todettiin, opinnäytetyön tuloksena saatiin luotua toimiva ja testattu tekoäly yhdelle vihollishahmotyypille. Tekoälyä ja animaatioista vastaavia luokkia pystytään käyttämään myös muilla hahmoilla, mutta tarvittavan grafiikan puuttumisen vuoksi niitä ei päästy testaamaan. Tekoälyn käyttö muille hahmoille vaatii toimivuustestauksen ja mahdollisesti tarvittavan tyyppikohtaisen hienosäädön ohjelmakoodissa.

Asiakas harkitsi Corrupted Culturan julkaisua Windows Phone -käyttöjärjestelmien lisäksi myös muille mobiilikäyttöjärjestelmille. Tekoälyn ohjelmakoodin kääntäminen suoraan alustalle, joka ei tue XNA:ta voi olla haasteellista, sillä tekoäly hyödyntää niin paljon XNA-sovelluskehyksen ominaisuuksia. Ainoastaan hahmon liikkumisessa keskeisessä roolissa oleva polunhakualgoritmi pyrittiin toteuttamaan mahdollisimman XNA-riippumattomasti, ja sen käyttö myös muissa peleissä ja muilla alustoilla onnistuu.

Tekoäly pyrittiin rakentamaan mahdollisimman modulaarisesti. Näin asiakas pystyy itse lisäämään kokonaisuuteen uusia komponentteja, mikäli Corrupted Culturan kehityksen myötä ilmenee lisää tekoälyltä vaadittuja ominaisuuksia.

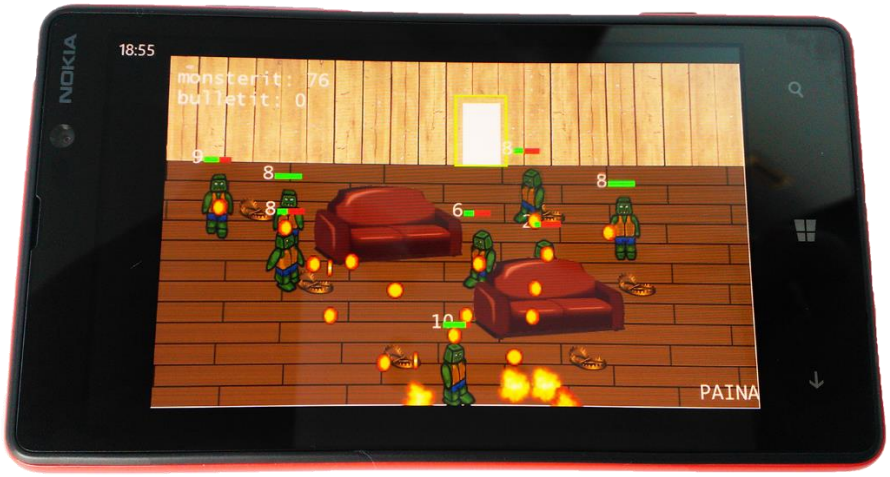
Oppimiskokemukset

Opinnäytetyön merkittävimmät käsitteet, tekoäly ja XNA, olivat ennestään lähes täysin tuntemattomia asioita. Toki pelitekoäly oli tuttu tietokonepeleistä vihollishahmoja ohjaavana tekijänä, mutta käytännön toteutusmahdollisuudet ja siihen liittyvä teoreettinen tieto olivat täysin uutta. XNA:sta puolestaan olin kuullut vain mainittavan Silverlightin ohella. Opeteltavaa siis riitti ja ennen varsinaisen työn aloittamista käytin aikaa etenkin XNA-sovelluskehyksen tarjoamiin mahdollisuuksiin tutustuen. Itse koin XNA:n helpoksi sisäistää, ja ohjelmoinnista XNA-sovelluskehystä hyödyntäen tuli melko nopeasti sujuvaa.

Tarkasteltaessa koko sovelluskehitysprojektia näin jälkikäteen, XNA:n opettelun sijaan projektin alussa olisi pitänyt painottaa enemmän tutustumista pelitekoälyn teoriaan sekä yleisimpiin toteutustekniikoihin. Näin ratkaisuissa olisi voinut hyödyntää jo olemassa olevia, toimivia malleja. Törmäsin useamman kerran tilanteeseen, jossa tekoälyä käsittelevästä teoksesta löytyi lähes samanlainen ratkaisu kuin minkä olin itse tehnyt. Esimerkiksi tilakoneen suunnittelussa teoriatiedon hyödyntäminen olisi varmasti johtanut parempaan lopputulokseen. Lisäksi valmiiden toteutustekniikoiden hyödyntäminen olisi helpottanut ratkaisujen suunnittelua ja nopeuttanut työn valmistumista.

Pidän arvokkaana kokemuksena sitä, että pääsin seuraamaan hyvinkin läheltä yrityksen perustamista. Aloittaessani opinnäytetyötä Jontka oli ottamassa vasta ensi askeliaan pelialan yrityksenä, joten oli mielenkiintoista nähdä, millaisia vaiheita yritystoiminnan aloittamiseen kuulu ja mitä asioita siinä on huomioitava.

Kaiken kaikkiaan projekti oli erittäin mielenkiintoinen ja tarjosi sopivasti haasteita. Projekti opetti paljon uutta ja tarjosi hyvät mahdollisuudet omien taitojen kehittämiseen ja uusien tekniikoiden opetteluun. Lisäksi projektin aikana pääsi soveltamaan opiskelun aikana opittuja tietoja ja taitoja käytännössä. Itselleni opinnäytetyön aihe oli mieleinen, sillä olin päättänyt, että haluan tehdä opinnäytetyön mobiilisovelluskehityksen parissa.



Kuva 32. Opinnäytetyönä suunniteltu ja toteutettu tekoäly toiminnassa

Kuvat

Taulukot

Lähteet

AppCampus 2013. Founding and coaching for Windows Phone developers. http://www.appcampus.fi/about/appcampus. Luettu 23.2.2013.

Dawes, A. 2010. Windows Phone 7 Game Development. Apress.

GamesIndustry International 2012. The State of Mobile Game Development. http://www.gamesindustry.biz/articles/2012-11-28-the-state-of-mobile-game-development. Luettu 31.3.2013.

Jaegers, K. 2010. XNA 4.0 Game Development by Example: Beginner's Guide. Packt Pub.

Jontka 2013. Corrupted Culturan suunnitteludokumentti.

Kirby, N. 2011. Introduction to Game AI. Cengage Learning.

Kokkarinen, I. 2003. Tekoäly, laskettavuus ja logiikka. Talentum Media Oy.

Lappeenranta Business & Innovations 2012. http://www.epressi.com/tiedotteet/paauutinen/uusi-tuotekehitystila-pelialan-yrityksille-skinnarilaan. Luettu 22.2.2013.

Lecky-Thompson, G. 2008. AI and Artificial Life in Video Games. Cengage Learning.

Microsoft 2013. Windows Phone SDK 8.0. http://www.microsoft.com/en-us/download/details.aspx?id=35471#overview. Luetuu 15.3.2013

Microsoft BizSpark 2013. BizSpark-ohjelma startup-yrityksille. http://www.microsoft.com/finland/bizspark/default.htm. Luettu 23.2.2013.

Microsoft Developer Network 2013. Application Lifecycle. http://msdn.microsoft.com/en-us/windowsphonetrainingcourse_applicationlifetimewp7lab.aspx. Luettu 2.3.2013.

Microsoft Developer Network 2013. Getting Started with Visual Studio. http://msdn.microsoft.com/en-us/library/ms165079.aspx. Luettu 15.3.2013.

Microsoft Developer Network 2013. Introducing the Windows Phone Application Life Cycle. http://msdn.microsoft.com/en-us/windowsphonetrainingcourse_applicationlifetimewp7lab_topic2#_Toc306869888. Luettu 2.3.2013.

Microsoft Developer Network 2012. Isolated Storage Overview for Windows Phone. http://msdn.microsoft.com/en-us/library/ff402541%28v=vs.92%29.aspx. Luettu 3.3.2013.

Microsoft Developer Network 2013. Registration info. http://msdn.microsoft.com/library/windowsphone/help/jj206719%28v=vs.105%29.aspx. Luettu 30.3.2013

Microsoft Developer Network 2013. Submit your app. http://msdn.microsoft.com/library/windowsphone/help/jj206724%28v=vs.105%29.aspx. Luettu 30.3.2013.

Microsoft Developer Network 2013. What is a Game Loop? http://msdn.microsoft.com/en-us/library/bb203873%28v=XNAGameStudio.40%29.aspx. Luettu 3.3.2013.

Miller, T. & Johnson, D. 2011. XNA Game Studio 4.0 Programming, Developing for Windows Phone 7 and Xbox 360. Addison Wesley.

Millington, I. & Funge, J. 2009. Artifical Intelligence for Games. CRC Press.

Molen, B. 2012. Windows Phone 8 review. Engadget. http://www.engadget.com/2012/10/29/windows-phone-8-review/. Luettu 14.3.2013.

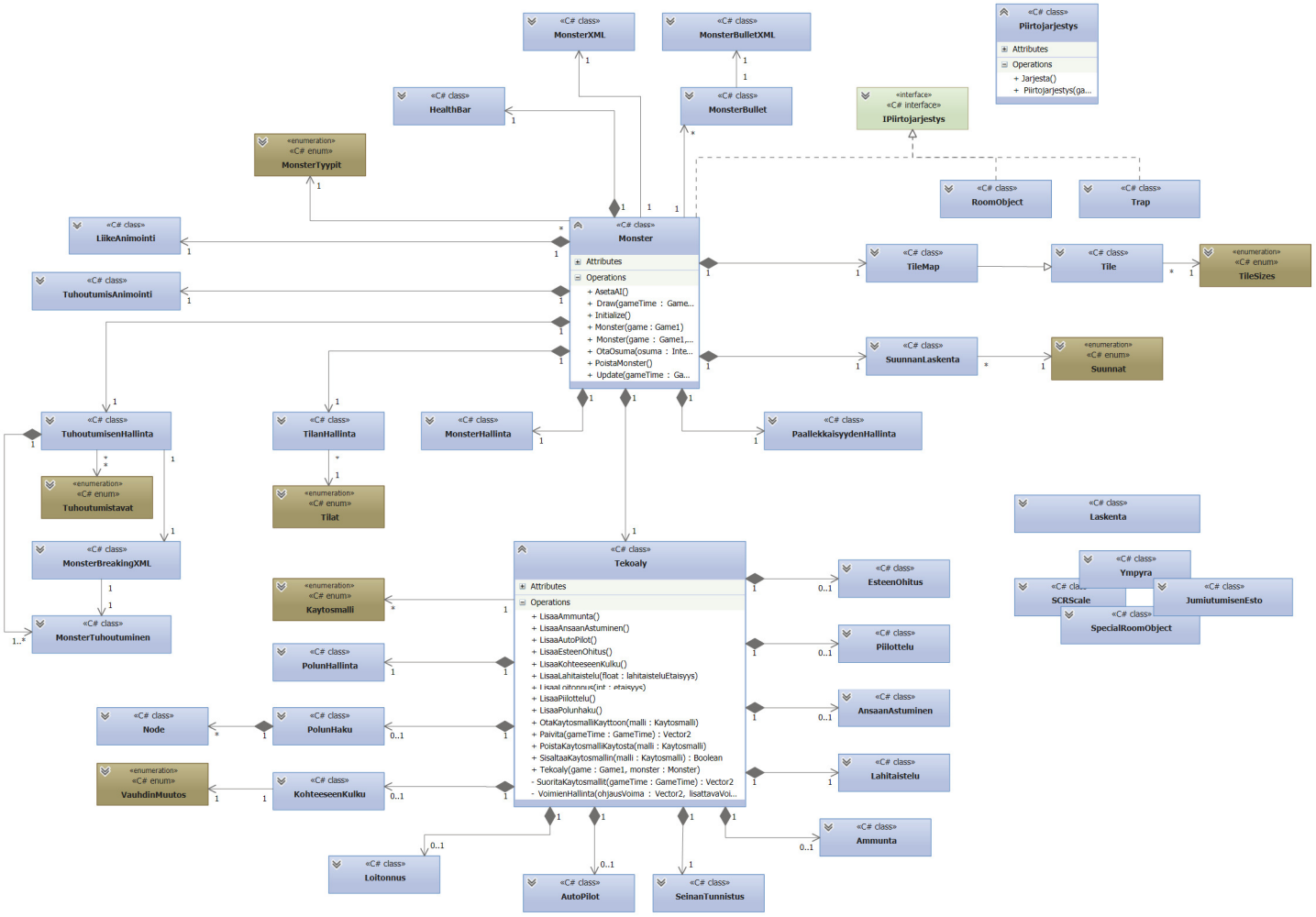
Nitschke, B. 2007. Professional XNA Game Programming – For Xbox 360 and Windows. Wiley Publishing.

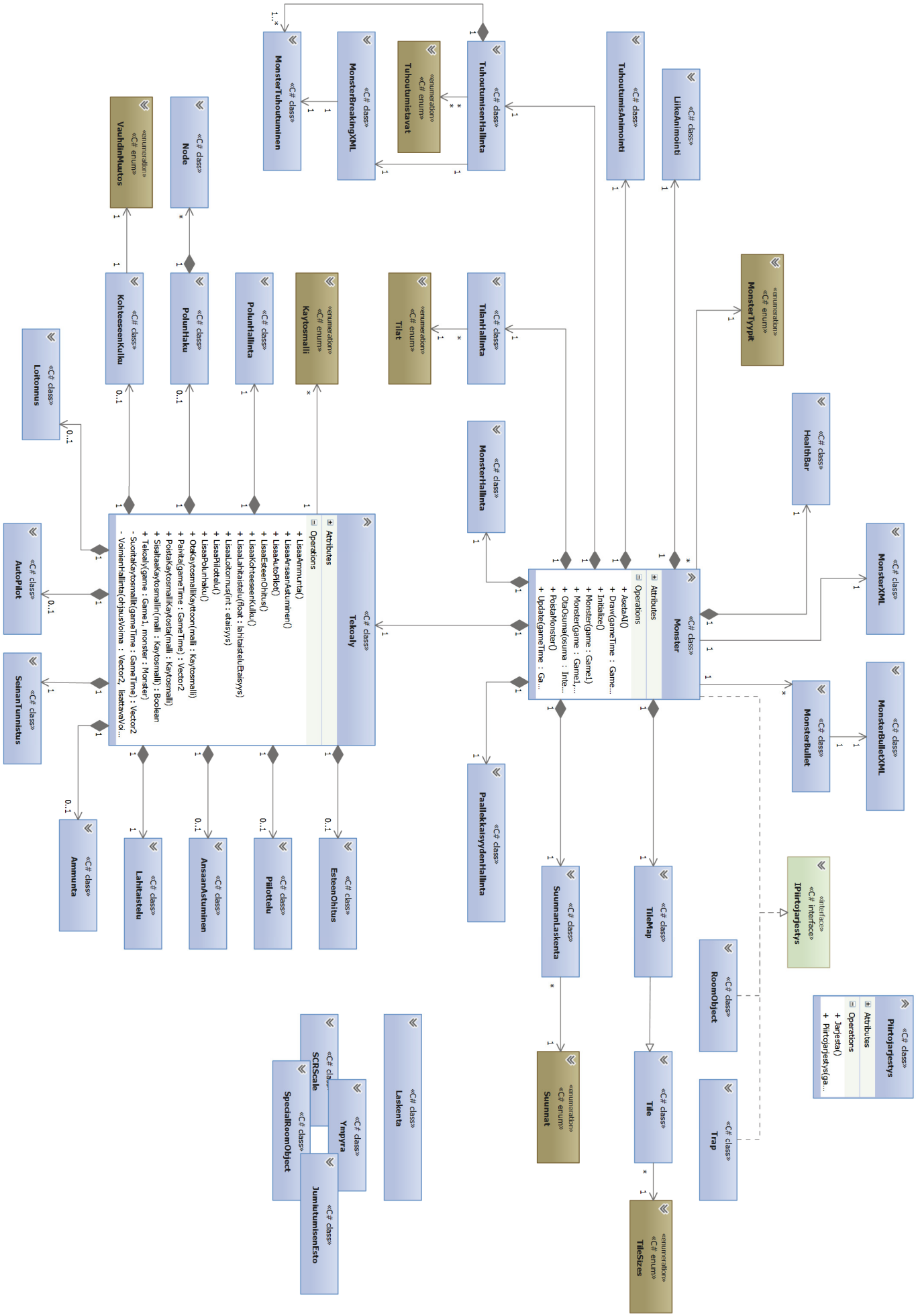
Petzold, C. 2010. Programming Windows Phone 7. Microsoft Press.

Rabin, S. 2002. AI Game Programming Wisdom. Charles River Media

Tekes 2012. Tekes valmistelee ohjelmaa pelialalle. http://www.tekes.fi/fi/community/Uutiset/404/Uutinen/1325?name=Tekes+valmistelee+ohjelmaa+pelialalle. Luettu 31.3.2013.

Ziegler, C. 2010. Windows Phone 7: the complete guide. Engadget. http://www.engadget.com/2010/03/18/windows-phone-7-series-the-complete-guide/. Luettu 14.3.2013.


«C# class»
MonsterXML
«C# class»
MonsterBulletXML
«C# class»
Piirtojarjestys
Attributes
Operations
+ Jarjesta()
+ Piirtojarjestys(ga...
«C# class»
HealthBar
«C# class»
MonsterBullet
«interface»
«C# interface»
IPiirtojarjestys
«enumeration»
«C# enum»
MonsterTyypit
«C# class»
RoomObject
«C# class»
Trap
«C# class»
LiikeAnimointi
«C# class»
Monster
Attributes
Operations
+ AsetaAI()
+ Draw(gameTime : Game...
+ Initialize()
+ Monster(game : Game1)
+ Monster(game : Game1,...
+ OtaOsuma(osuma : Inte...
+ PoistaMonster()
+ Update(gameTime : Ga...
«C# class»
TileMap
«C# class»
Tile
«enumeration»
«C# enum»
TileSizes
«C# class»
TuhoutumisAnimointi
«C# class»
SuunnanLaskenta
«enumeration»
«C# enum»
Suunnat
«C# class»
TuhoutumisenHallinta
«C# class»
TilanHallinta
«C# class»
MonsterHallinta
«C# class»
PaallekkaisyydenHallinta
«enumeration»
«C# enum»
Tuhoutumistavat
«enumeration»
«C# enum»
Tilat
«C# class»
Laskenta
«C# class»
MonsterBreakingXML
«C# class»
Tekoaly
Attributes
Operations
+ LisaaAmmunta()
+ LisaaAnsaanAstuminen()
+ LisaaAutoPilot()
+ LisaaEsteenOhitus()
+ LisaaKohteeseenKulku()
+ LisaaLahitaistelu(float : lahitaisteluEtaisyys)
+ LisaaPiirtonro(int : etaisyys)
+ LisaaPiilottelu()
+ LisaaPolunhaku()
+ OtaKaytosmalliKayttoon(malli : Kaytosmalli)
+ Paivita(gameTime : GameTime) : Vector2
+ PoistaKaytosmalliKaytosta(malli : Kaytosmalli)
+ SisaltaaKaytosmallin(malli : Kaytosmalli) : Boolean
+ Tekoaly(game : Game1, monster : Monster)
- SuoritaKaytosmallit(gameTime : GameTime) : Vector2
- VoimienHallinta(ohjausVoima : Vector2, lisattavaVoi...
«C# class»
Ympyra
«C# class»
SCRScale
«C# class»
JumiutumisenEsto
«C# class»
SpecialRoomObject
«C# class»
MonsterTuhoutuminen
«enumeration»
«C# enum»
Kaytosmalli
«C# class»
EsteenOhitus
«C# class»
PolunHallinta
«C# class»
Piilottelu
«C# class»
Node
«C# class»
PolunHaku
«C# class»
AnsaanAstuminen
«enumeration»
«C# enum»
VauhdinMuutos
«C# class»
KohteeseenKulku
«C# class»
Lahitaistelu
«C# class»
Ammunta
«C# class»
Loitonnus
«C# class»
AutoPilot
«C# class»
SeinanTunnistus


«C# class»
Monster
Attributes
Operations
+ AsetaAI()
+ Draw(gameTime : Game...
+ Initialize()
+ Monster(game : Game1)
+ Monster(game : Game1,...
+ OtaOsuma(osuma : Inte...
+ PoistaMonster()
+ Update(gameTime : Ga...
«C# class»
Tekoaly
Attributes
Operations
+ LisaaAmmunta()
+ LisaaAnsaanAstuminen()
+ LisaaAutoPilot()
+ LisaaEsteenOhitus()
+ LisaaKohteeseenKuluku()
+ LisaaLahitaistelu(float : lahitaistelunEtaisyys)
+ LisaaLoitonnus(int : etaisyys)
+ LisaaPiilotteli()
+ LisaaPolunHaku()
+ OtaKaytosmalliKayttoon(malli : Kaytosmalli)
+ PaivitaGameTime : GameTime) : Vector2
+ PoistaKaytosmalliKaytosta(malli : Kaytosmalli)
+ SisaltaaKaytosmallin(malli : Kaytosmalli) : Boolean
+ Tekoaly(game : Game1, monster : Monster)
+ TekoalyGameTime : GameTime) : Vector2
- SuoritaKaytosmalli(gameTime : GameTime) : Vector2
- VoinkoHallintaOhjausVoima : Vector2, IlisattavaVoi...
«C# class»
MonsterXML
«C# class»
MonsterBulletXML
«C# class»
MonsterBullet
«C# class»
HealthBar
«C# class»
MonsterHallinta
«C# class»
PaallekkaisyydenHallinta
«C# class»
SuunnanLaskenta
«enumeration»
«C# enum»
Suunnat
«C# class»
TileMap
«C# class»
Tile
«enumeration»
«C# enum»
TileSizes
«C# class»
RoomObject
«C# class»
Trap
«interface»
«C# interface»
IPiirtojarjestys
«C# class»
PiirtojarjestysAttributes
Operations
+ Jarjesta()
+ Piirtojarjestys(ga...
«enumeration»
«C# enum»
MonsterTyypit
«C# class»
LikeAnimointi
«C# class»
TuhoutumisAnimointi
«C# class»
TuhoutumisenHallinta
«enumeration»
«C# enum»
Tuhoutumistavat
«C# class»
MonsterBreakingXML
«C# class»
MonsterTuhoutuminen
«C# class»
TilanHallinta
«enumeration»
«C# enum»
Tilat
«C# class»
PolunHallinta
«C# class»
PolunHaku
«C# class»
KohteeseenKuluku
«enumeration»
«C# enum»
Kaytosmalli
«enumeration»
«C# enum»
VauhdinMuutos
«C# class»
Node
«C# class»
Loitonnus
«C# class»
AutoPilot
«C# class»
SeinanTunnistus
«C# class»
Ammunta
«C# class»
Lahitaistelu
«C# class»
AnsaanAstuminen
«C# class»
Piilottelu
«C# class»
EsteenOhitus
«C# class»
Laskenta
«C# class»
SCRScale
«C# class»
SpecialRoomObject
«C# class»
Ympyra
«C# class»
JumiutumisenEsto

Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Tietojärjestelmien kehitys

Sami Anttonen

Tekoälyn suunnittelu ja toteutus mobiilipeliin

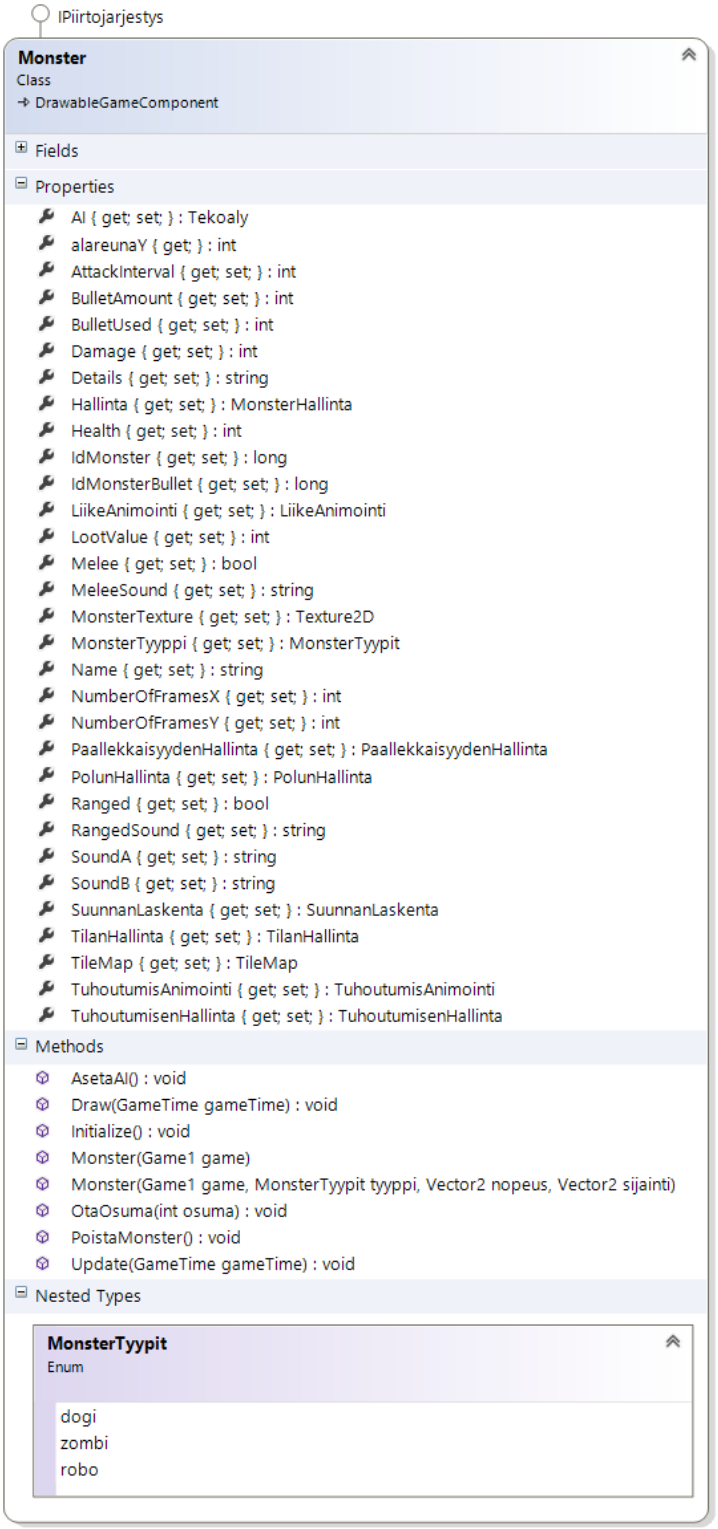
Liite 2: Tekoälyn keskeisimmät luokat

Opinnäytetyön liite, 2013

Sisältö

1 Hahmo

1.1 Monster-luokka



```
15  namespace CCMonsters
16  {
17      //MONSTER (vihollishahmo)
18      //Luokka perii XNA:n DrawableGameComponent-luokan
19      //ja toteuttaa IPiirtojarjestys-rajapinnan
20
21      public class Monster : Microsoft.Xna.Framework.DrawableGameComponent, IPiirtojarjestys
22      {
23          Game1 game;
24          SpriteBatch spriteBatch;
25          HealthBar healthBar;
26
27          //Mahdolliset monstertyypit
28          public enum MonsterTyypit
29          {
30              dogi,
31              zombi,
32              robo
33          }
34
35          PERUSMUUTTUJAT (luetaan xml-tiedostosta)
167
168         ANIMAATIOT (muuttujat + ominaisuudet)
185
186         TEKOÄLY ja HALLINTA (muuttujat + ominaisuudet)
253
254
255         public Monster(Game1 game)
256             : base(game) { }
257
258         public Monster(Game1 game, MonsterTyypit tyyppi, Vector2 nopeus, Vector2 sijainti)
259             : base(game)
260         {
261             this.game = game;
262             this.monsterTyyppi = tyyppi;
263             spriteBatch = new SpriteBatch(game.GraphicsDevice);
264             this.healthBar = new HealthBar(game, this);
265
266             this.hallinta = new MonsterHallinta();
267
268             //Monsterin tietojen asetus XML-tiedostosta lukemalla
269             MonsterXML monsterXML = new MonsterXML(game);
270             switch (tyyppi)
271             {
272                 case (MonsterTyypit.zombi):
273                     monsterXML.AsetaXmlTiedot(this, 2);
274                     hallinta.RajaSade = 15;
275                     this.liikeAnimointi = new LiikeAnimointi(game, this, 65);
276                     break;
277                 case (MonsterTyypit.robo):
278                     monsterXML.AsetaXmlTiedot(this, 1);
279                     hallinta.RajaSade = 40f;
280                     this.liikeAnimointi = new LiikeAnimointi(game, this, 40);
281                     break;
282                 case (MonsterTyypit.dogi):
283                     monsterXML.AsetaXmlTiedot(this, 4);
284                     hallinta.RajaSade = 20f;
285                     this.liikeAnimointi = new LiikeAnimointi(game, this, 40);
286                     break;
287             }
```

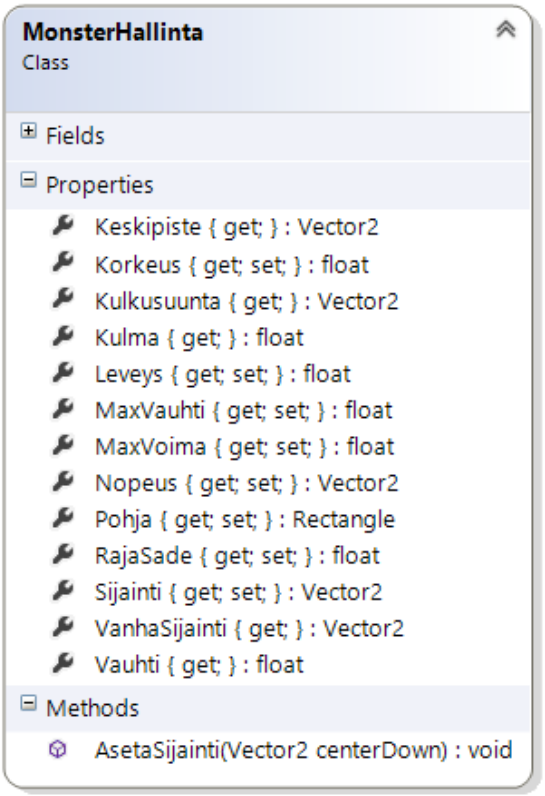
```
288
289             this.tuhoutumisAnimointi = new TuhoutumisAnimointi(game, this);
290             this.tilanHallinta = new TilanHallinta(game, this);
291             this.suunnanLaskenta = new SuunnanLaskenta(game, this);
292
293             this.hallinta.Nopeus = nopeus;
294             this.hallinta.Sijainti = sijainti;
295
296             paallekkaisyydenHallinta = new PaallekkaisyydenHallinta(game, this);
297             tuhoutumisenHallinta = new TuhoutumisenHallinta(game, this);
298
299             AsetaAI();
300         }
301
302         public override void Initialize()
303         {
304             base.Initialize();
305         }
306
307         public override void Update(GameTime gameTime)
308         {
309             if (TilanHallinta.Tila != TilanHallinta.Tilat.Ansassa)
310             {
311                 ai.PathManager.Paivita();
312             }
313
314             paallekkaisyydenHallinta.ValtaPaallekkaisyydet();
315
316             Hallinta.Nopeus += AI.Paivita(gameTime);
317
318             SuunnanLaskenta.LaskeSuunta(gameTime);
319             TilanHallinta.Paivita(gameTime);
320
321             if (TilanHallinta.Tila != TilanHallinta.Tilat.Ansassa)
322             {
323                 Hallinta.Sijainti += Vector2.Normalize(Hallinta.Nopeus) * 1.8f;
324             }
325
326             base.Update(gameTime);
327         }
328
329         public override void Draw(GameTime gameTime)
330         {
331             spriteBatch.Begin();
332             SpriteFont font = game.Content.Load<SpriteFont>("font");
333             spriteBatch.DrawString(font, bulletUsed.ToString(),
334                 new Vector2(Hallinta.Sijainti.X, Hallinta.Sijainti.Y - 40), Color.White);
335
336             //Kuljettavan polun piirto
337             //Texture2D texture = game.Content.Load<Texture2D>("Picture/PathPoint");
338             //foreach (Vector2 p in ai.Polunhaku.Polku)
339             //{
340             //    spriteBatch.Draw(texture, p, Color.Red);
341             //}
342             spriteBatch.End();
343
344             //ANIMAATIOT
345             switch (tilanHallinta.Tila)
346             {
```

```
347                 case (TilanHallinta.Tilat.Sisaantulo):
348                     liikeAnimointi.Sisaantulo(gameTime);
349                     break;
350                 case (TilanHallinta.Tilat.AutoPilot):
351                 case (TilanHallinta.Tilat.Piiloutuu):
352                 case (TilanHallinta.Tilat.Piilossa):
353                 case (TilanHallinta.Tilat.PoistuuPiilosta):
354                 case (TilanHallinta.Tilat.Jumissa):
355                 case (TilanHallinta.Tilat.PolunHaku):
356                 case (TilanHallinta.Tilat.KiertaaAnsaa):
357                     healthBar.OtaKayttoon();
358                     switch (suunnanLaskenta.Suunta)
359                     {
360                         case (SuunnanLaskenta.Suunnat.Suoraan):
361                             liikeAnimointi.LiikkuSuoraan(gameTime);
362                             break;
363                         case (SuunnanLaskenta.Suunnat.Vasemmalle):
364                             liikeAnimointi.LiikkuVasemmalle(gameTime);
365                             break;
366                         case (SuunnanLaskenta.Suunnat.Oikealle):
367                             liikeAnimointi.LiikkuOikealle(gameTime);
368                             break;
369                     }
370                     break;
371                 case (TilanHallinta.Tilat.Ansassa):
372                 case (TilanHallinta.Tilat.OttaaOsumaa):
373                     healthBar.OtaKayttoon();
374                     switch (suunnanLaskenta.Suunta)
375                     {
376                         case (SuunnanLaskenta.Suunnat.Vasemmalle):
377                             tuhoutumisAnimointi.OsumaAnimaatioVasemalle(gameTime,
378                                 tuhoutumisenHallinta.ValitseTuhoutuminen());
379                             break;
380                         case (SuunnanLaskenta.Suunnat.Oikealle):
381                             tuhoutumisAnimointi.OsumaAnimaatioOikealle(gameTime,
382                                 tuhoutumisenHallinta.ValitseTuhoutuminen());
383                             break;
384                         case (SuunnanLaskenta.Suunnat.Suoraan):
385                             tuhoutumisAnimointi.OsumaAnimaatioVasemalle(gameTime,
386                                 tuhoutumisenHallinta.ValitseTuhoutuminen());
387                             break;
388                     }
389                     break;
390 
391                 case (TilanHallinta.Tilat.Kuolee):
392                     healthBar.PoistaKaytosta();
393 
394                     switch (tuhoutumisenHallinta.Tuhoutumistapa)
395                     {
396                         case (TuhoutumisenHallinta.Tuhoutumistavat.Oletus):
397                             if (tuhoutumisAnimointi.KuolinAnimaatio(gameTime,
398                                 tuhoutumisenHallinta.ValitseTuhoutuminen()))
399                             {
400                                 PoistaMonster();
401                             }
402                             break;
403                     }
404                     break;
405             }
406         }
```

```
407
408         //Tuhotaan monster-olio ja vapautetaan sen varaamat resurssit
409         public void PoistaMonster()
410         {
411             if (Ranged)
412             {
413                 if (AI != null)
414                 {
415                     if (AI.SisaltaaKaytosmallin(Tekoaly.Kaytosmalli.piilottelu))
416                     {
417                         AI.Piilottelu.PoistaPiilottelija(this);
418                     }
419                 }
420             }
421             game.monsters.Remove(this);
422             game.Components.Remove(this);
423             game.piirrettavat.Remove(this);
424             this.Dispose();
425             game.tuhotutMonsterit++;
426         }
427
428         //Vähentää monsterin elämää parametrina saatavan osuman verran
429         public void OtaOsuma(int osuma)
430         {
431             Health = Health - osuma;
432             //Vector2 vektoriPelaajaan = new Vector2(400, 460) - Hallinta.Keskipiste;
433             Hallinta.Sijainti += Vector2.Normalize(Hallinta.Nopeus) * -10;
434         }
435
436         //Asettaa monsterille tekoälyn tyyppikohtaisesti
437         public void AsetaAI()
438         {
439             switch (monsterTyyppi)
440             {
441                 //TYYPPI = ZOMBI
442                 #region ZOMBI
443                 case (MonsterTyypit.zombi):
444                     tileMap = new TileMap(game, monsterTyyppi, Tile.TileSizes.zombi);
445                     ai = new Tekoaly(game, this);
446
447                     ai.LisaaLoitonnus(50);
448                     ai.LisaaKohteeseenKulku();
449                     ai.LisaaAnsaanAstuminen();
450                     ai.LisaaPolunhaku();
451                     ai.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.polunHaku);
452                     ai.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.kohteeseenKulku);
453                     ai.LisaaLahitaistelu(50);
454                     if (Ranged)
455                     {
456                         ai.LisaaPiilottelu();
457                         ai.LisaaAmmunta();
458                         ai.LisaaAutoPilot();
459                     }
460                     break;
461                 #endregion
462
463                 //TYYPPI = ROBO
464                 ROBOTTI
```

```
484
485                 //TYYPPI = DOGI
486                 KOIRA
498             }
499         }
500
501         //Toteutetaan IPiirtojarjestys-rajapinta
502         public int alareunaY
503         {
504             get
505             {
506                 return (int)(this.Hallinta.Sijainti.Y + this.Hallinta.Korkeus);
507             }
508         }
509     }
510 }
511
```

1.2 Hahmon etenemisen hallinta




```
13 namespace CCMonsters
14 {
15     //Luokka monsterin hallintaa varten
16     //Pitää muuttujissa tiedot monsterin etenemisestä pelimaailmassa
17
18     public class MonsterHallinta
19     {
20         MUUTTUJAT
36
37         #region OMINAISUUDET
```

```
38
39          //Kulma, jossa monsteri etenee
40          public float Kulma
41          {
42              get { return Laskenta.LaskeKulmaRadiaaneina(vanhaSijainti, sijainti); }
43          }
44
45          public float Leveys
46          {
47              get { return leveys; }
48              set { leveys = value; }
49          }
50
51          public float Korkeus
52          {
53              get { return korkeus; }
54              set { korkeus = value; }
55          }
56
57          //Monsterille (keskipisteeseen) asetettu säde
58          public float RajaSade
59          {
60              get { return rajaSade; }
61              set { rajaSade = value; }
62          }
63
64          //Nopeus-vektori (etenemiseen liittyvä suunta ja suuruus)
65          public Vector2 Nopeus
66          {
67              get { return nopeus; }
68              set { nopeus = value; }
69          }
70
71          //Monsterin etenemissuuntaan asetettu vektori
72          public Vector2 Kulkusuunta
73          {
74              get
75              {
76                  return new Vector2(
77                      (float)Math.Abs(Math.Cos(this.Kulma)),
78                      (float)Math.Abs(Math.Sin(this.Kulma)));
79              }
80          }
81
82          //Vauhti, jolla monsteri etenee (Nopeuden suuruus)
83          public float Vauhti
84          {
85              get { return Nopeus.Length(); }
86          }
87
88          //Sijainti, jossa monsteri on (vasemman yläkulman piste)
89          public Vector2 Sijainti
90          {
91              get { return sijainti; }
92              set
93              {
94                  vanhaSijainti = sijainti;
95                  sijainti = value;
96              }
97          }
```

```
98
99          //Sijainti, jossa monsteri olli edellisen päivityksen yhteydessä
100         public Vector2 VanhaSijainti
101         {
102             get { return vanhaSijainti; }
103         }
104
105         //Monsterin (spriten) keskipiste
106         public Vector2 Keskipiste
107         {
108             get
109             {
110                 return new Vector2(Sijainti.X + Leveys / 2, Sijainti.Y + Korkeus / 2);
111             }
112         }
113
114         //Suurin mahdollinen vauhti
115         public float MaxVauhti
116         {
117             get { return maxVauhti; }
118             set { maxVauhti = value; }
119         }
120
121         //Suurin mahdollinen voima
122         public float MaxVoima
123         {
124             get { return maxVoima; }
125             set { maxVoima = value; }
126         }
127
128         //Monsterin jalkojen viemä ala (pitänee lisätä jossain vaiheessa myös kantaan)
129         //Käytetään asetettaessa monstereita tilemapiin (vrt. esteet tilemapissa)
130         public Rectangle Pohja
131         {
132             get { return new Rectangle((int)Sijainti.X - 10,
133                 (int)Sijainti.Y + (int)Korkeus - 70, (int)Leveys + 20, 60); }
134             set { pohja = value; }
135         }
136
137         #endregion
138
139         METODIT
151     }
152 }
153
```

2 Piirtojärjestys

2.1 IPiirtojarjestys-rajapinta

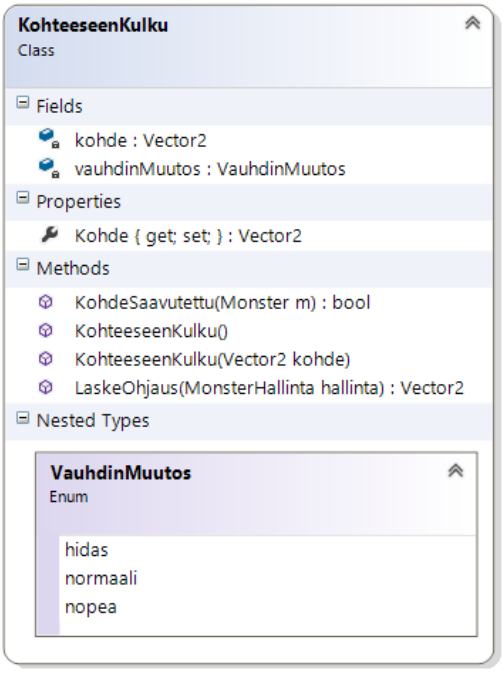
```
4  namespace CCMonsters.CCMonster
5  {
6      //IPiirtojarjestys-rajapinta
7      //Kaikkien kohteiden, joiden piirtojärjestystä halutaan hallita,
8      //täytyy toteuttaa tämä rajapinta
9
10     public interface IPiirtojarjestys
11     {
12         //ominaisuuten kohteen alareunan y-koordinaatti,
13         //joka voidaan vain palauttaa (ei voi asettaa)
14         int alareunaY { get;}
15     }
16 }
17
```

2.2 Piirtojarjestys-luokka

```
14 namespace CCMonsters
15 {
16     //Piirtojärjestyksen hallinta
17     //Piirtojarjestys-luokka vastaa näytölle piirrettävien kohteiden piirtojärjestyksestä
18     //eli DrawOrder-arvon asettamisesta.
19
20     public class Piirtojarjestys
21     {
22         Game1 game;
23         List<IPiirtojarjestys> lajiteltavaLista;
24
25         public Piirtojarjestys(Game1 game)
26         {
27             this.game = game;
28             this.lajiteltavaLista = new List<IPiirtojarjestys>();
29         }
```

```
30          //Jarjesta-metodi lajittelee IPiirtojärjestys-rajapinnan toteuttavien olioiden
31          //listan y-koordinaatin mukaan, ja asettaa DrawOrder-arvon.
32          //Metodia kannattaa kutsua esim. pelin pääluokan Update-metodin yhteydessä.
33          public void Jarjesta()
34          {
35              this.lajiteltavaLista = game.piirrettavat;
36
37              //Lajitellaan lista y-koordinaattien mukaan pienimmästä suurimpaan
38              lajiteltavaLista.Sort((x, y) => x.alareunaY.CompareTo(y.alareunaY));
39
40              //Asetetaan DrawOrder sen mukaan mikä kohteen paikka lajitellussa listassa on
41              //Mitä pienempi y-koordinaatti sitä pienempi DrawOrder arvo
42              //--> pienemmän arvon omaava piirtyy alle
43              //Käytössä DrawOrder-arvoksi luvut jotka ovat suurempia kuin 100,
44              //koska 0-100 on varattuna taustan piirtoa varten
45              for (int i = 0; i < lajiteltavaLista.Count; i++)
46              {
47                  if (lajiteltavaLista[i] is Monster)
48                  {
49                      ((Monster)lajiteltavaLista[i]).DrawOrder = i + 101;
50                  }
51                  else if ((lajiteltavaLista[i] is RoomObject))
52                  {
53                      ((RoomObject)lajiteltavaLista[i]).DrawOrder = i + 101;
54                  }
55              }
56          }
57      }
58  }
59
```

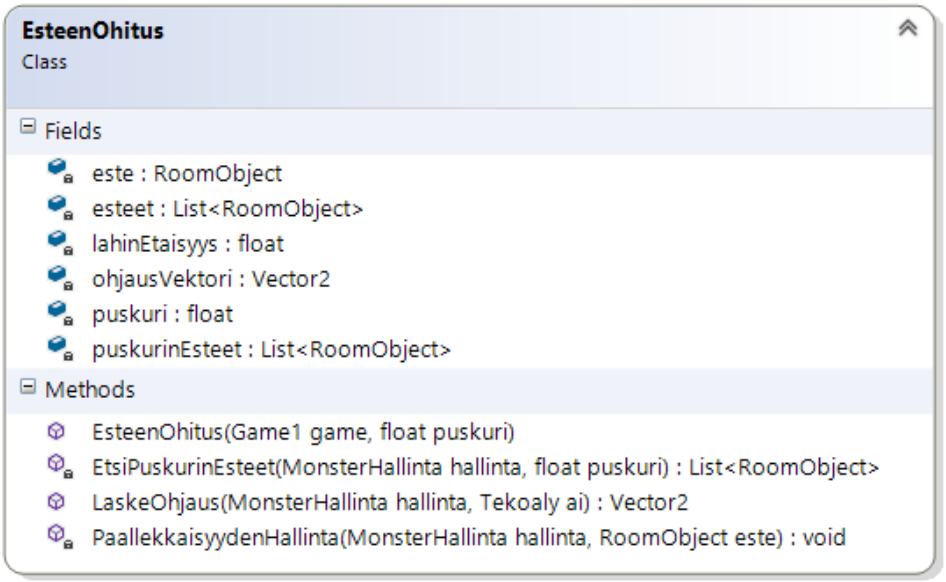
3 Kohteeseen kulku



```
13  namespace CCMonsters
14  {
15      //KOHTEESEEN KULKU
16      //KohteeseenKulku-luokan tehtävänä on selvittää tarvittava ohjaus, jotta monsteri
17      //kulkee kohti kohdetta
18      //LaskeOhjaus-metodi palauttaa tarvittavaa ohjausta vastaavan vektorin
19
20      public class KohteeseenKulku
21      {
22          //kohde, johon täytyy päästä
23          Vector2 kohde;
24          public Vector2 Kohde
25          {
26              get { return kohde; }
27              set { kohde = value; }
28          }
29
30          //Kuinka nopeasti monsterin liike hidastuu
31          public enum VauhdinMuutos
32          {
33              hidas = 3,
34              normaali = 2,
35              nopea = 1,
36          };
37          private VauhdinMuutos vauhdinMuutos = VauhdinMuutos.normaali;
38
39          public KohteeseenKulku()
40          {
41              this.kohde = Vector2.Zero;
42          }
43          public KohteeseenKulku(Vector2 kohde)
44          {
45              this.kohde = kohde;
46          }
47
48          public Vector2 LaskeOhjaus(MonsterHallinta hallinta)
49          {
50              //Nopeus (haluttu suunta), jonka mukaan monsterin täytyy kulkea,
51              //jotta se saavuttaa kohteen
52              Vector2 haluttuNopeus = new Vector2();
53
54              //Tarkastetaan, että kohde on asetettu
55              if (kohde != null)
56              {
57                  //Vectori monsterista kohteeseen
58                  Vector2 kohteeseen = kohde - hallinta.Keskipiste;
59
60                  // Etäisyys kohteeseen
61                  float etaisyys = kohteeseen.Length();
62
63                  //Niin kauan kun etäisyys kohteeseen on suurempi kuin 0,
64                  //lasketaan monsterille uusi ohjaus eli uusi nopeus.
65                  if (etaisyys > 0)
66                  {
67                      //Kerroin, jolla hidastumista voidaan tarvittaessa hienosäätää
68                      const float muutosKerroin = 20f;
69
70                      //Kuvaa vauhtia, jolla monsterin pitäisi edetä
71                      //Mitä pienempi etäisyys kohteeseen on,
72                      //sitä pienemmän arvon muuttuja saa
73                      //Huomioidaan kuinka nopeasti vauhdin täytyy muuttua
74                      float vauhti = etaisyys / ((float)vauhdinMuutos * muutosKerroin);
```

```
75
76                      //Varmistetaan, että vauhti ei ylitä monsterille asetettua
77                      //suurinta mahdollista vauhtia
78                      vauhti = MathHelper.Min(vauhti, hallinta.MaxVauhti);
79
80                      //Lasketaan haluttuNopeus-vektori suhteuttamalla kohteeseen-vektori
81                      //vaadittuun vauhtiin
82                      haluttuNopeus = kohteeseen * vauhti / etaisyys;
83
84                      //Palautetaan ohjausvoima
85                      return (haluttuNopeus - hallinta.Nopeus);
86                  }
87              }
88              //Jos kohdetta ei ole asetettu,
89              //tai, jos monsteri on jo saavuttanut kohteen,
90              //palautetaan nollavektori, jolla ei ole vaikutusta monsterin liikkeeseen
91              return Vector2.Zero;
92          }
93
94          public bool KohdeSaavutettu(Monster m)
95          {
96              Vector2 etaisyys = m.Hallinta.Keskipiste - kohde;
97              if (etaisyys.Length() < 10)
98              {
99                  return true;
100             }
101             return false;
102         }
103     }
104 }
105
```

4 Esteen ohitus ympyrän kehää hyödyntäen

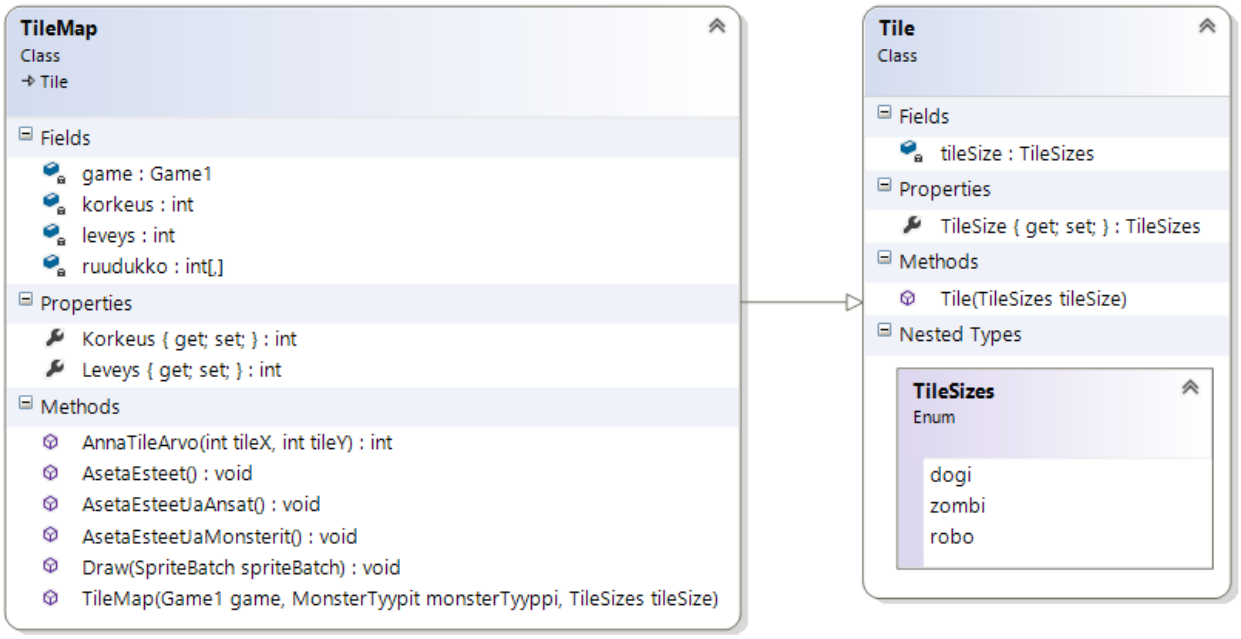


```
13  namespace CCMonsters
14  {
15      //ESTEEN OHITUS
16      //Käyttää esteen ohitukseen tekniikkaa, jossa hahmo kiertää esteen
17      //sen keskipisteeseen asetetun ympyrän kehää pitkin
18      public class EsteenOhitus
19      {
20          private float puskuri;
21          private float lahinEtaisyys;
22          private Vector2 ohjausVektori;
23          private List<RoomObject> esteet;
24          private List<RoomObject> puskurinEsteet;
25          private RoomObject este;
26
27          public EsteenOhitus(Game1 game, float puskuri)
28          {
29              this.puskuri = puskuri;
30              this.puskurinEsteet = new List<RoomObject>();
31              this.esteet = game.roomObjects;
32              this.lahinEtaisyys = int.MaxValue;
33          }
34
35          //Palauttaa listan puskurialueella olevista esteistä
36          private List<RoomObject> EtsiPuskurinEsteet(
37              MonsterHallinta hallinta, float puskuri)...
62
63          //Palauttaa vektorin, jonka mukaan kohdetta täytyy ohjata,
64          //jotta vältetään osuminen esteeseen
65          public Vector2 LaskeOhjaus(MonsterHallinta hallinta, Tekoaly ai)
66          {
67              if (puskurinEsteet != null)
68                  puskurinEsteet.Clear();
69
70              //Monsterin ja esteen välinen etäisyys
71              float etaisyysTormaykseen = 0.050f * hallinta.Vauhti;
72
73              //Esteen etäisyys suhteutettuna kohteen nopeuteen
74              float puskuriAlue = this.puskuri +
75                  (hallinta.Vauhti / hallinta.MaxVauhti) * this.puskuri;
76
77              //Etsitään mahdolliset esteet puskurialueen sisällä
78              //ja asetetaan ne muuttujaan esteet
79              List<RoomObject> esteet = EtsiPuskurinEsteet(hallinta, puskuriAlue);
80
81              this.ohjausVektori = Vector2.Zero;
82
83              //Käydään läpi vain puskurialueella olevat esteet
84              foreach (RoomObject ro in esteet)
85              {
86                  float etaisyysEsteenKeskustaan = etaisyysTormaykseen + ro.Sade;
87
88                  //kokonaisSade = esteen säde + monsterille kuviteltu säde
89                  float kokonaisSade = ro.Sade + hallinta.RajaSade;
90
91                  //Ensteen keskipiste suhteessa monsterin sijaintiin
92                  //Vektori a = vektori hahmon keskipisteestä esteen keskipisteeseen
93                  Vector2 vectorA = ro.Keskipiste - hallinta.Sijainti;
```

```
94
95                  //Kulkusuunta-vektorin ja a-vektorin välinen pistetulo
96                  float pistetulo = Vector2.Dot(vectorA, hallinta.Kulkusuunta);
97                  //Vektori x = vektori hahmon kulkusuunnan akselia myöten
98                  //esteen keskipisteen tasolle
99                  Vector2 vectorX = pistetulo * Vector2.Normalize(hallinta.Kulkusuunta);
100
101                 //Vektori c = Etäisyys/poikkeama edessä olevalta akselilta esteen keskustaan
102                 Vector2 vectorC = vectorA - vectorX;
103
104                 //Esteen kohtaamisen ehdot
105                 bool puskurialueella = vectorC.Length() < kokonaisSade;
106                 bool lahella = pistetulo < etaisyysEsteenKeskustaan;
107                 bool etupuolella = pistetulo > 0;
108
109                 //Jos ehdot täyttyvät siirrytään kauemmas esteen keskustasta
110                 if (puskurialueella && etupuolella && lahella)
111                 {
112                     float etaisyys = (vectorC * -1).Length();
113                     if (etaisyys < lahinEtaisyys)
114                     {
115                         lahinEtaisyys = etaisyys;
116                         ohjausVektori = vectorC * -1;
117                         this.este = ro;
118                     }
119                 }
120             }
121             this.PaallekkaisyydenHallinta(hallinta, this.este);
122             return ohjausVektori;
123         }
124         //Hoitaa mahdollisen päälekkäisyystilanteen esteen kanssa
125         private void PaallekkaisyydenHallinta(MonsterHallinta hallinta, RoomObject este)...
148     }
149 }
150
```

5 A*-polunhakualgoritmi

5.1 TileMap ja Tile



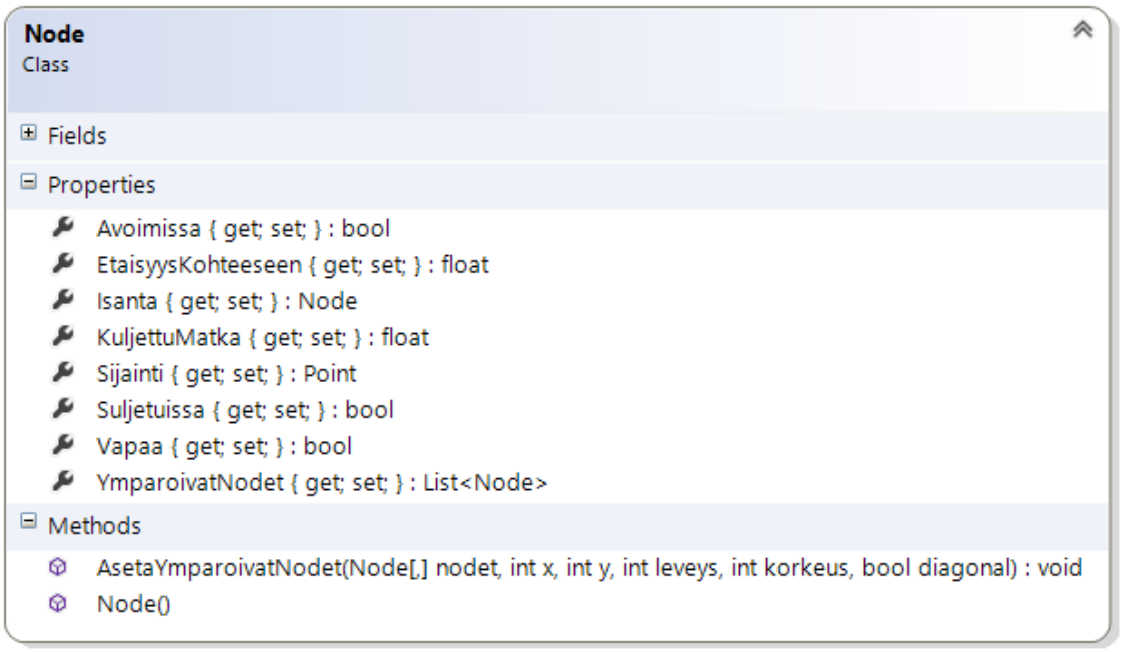
5.1.1 TileMap-luokka

```
13  namespace CCMonsters.CCMonster.Tekoaly.Polunhaku
14  {
15      //TILEMAP
16      //Tallentaa tiedon pelialueesta kaksiulotteiseen int-taulukkoon (ruudukko)
17      //Jos alkiota vastaava kohta pelialueella sisältää esteen, alkion arvoksi asetetaan 1,
18      //muuten arvo on 0
19
20      public class TileMap : Tile
21      {
22          Game1 game;
23
24          //kaksiuloitteinen taulukko, johon pelikentän osat tallennetaan
25          private int[,] ruudukko;
26          private int leveys;
27          private int korkeus;
28
29          public int Leveys
30          {
31              get { return leveys; }
32              set { leveys = value; }
33          }
34          public int Korkeus
35          {
36              get { return korkeus; }
37              set { korkeus = value; }
38          }
39
```

```
40          public TileMap(Game1 game, Monster.MonsterTyypit monsterTyyppi, TileSizes tileSize)
41              : base(tileSize)
42          {
43              this.game = game;
44
45              switch (monsterTyyppi)
46              {
47                      //alustetaan taulukko oikean kokoiseksi,
48                      //riippuen monsterin koosta
49                  case (Monster.MonsterTyypit.dogi):
50                      this.ruudukko = new int[36, 81];
51                      break;
52                  case (Monster.MonsterTyypit.zombi):
53                      this.ruudukko = new int[24, 54];
54                      break;
55                  case (Monster.MonsterTyypit.robo):
56                      this.ruudukko = new int[12, 27];
57                      break;
58              }
59
60              this.leveys = ruudukko.GetLength(1);
61              this.korkeus = ruudukko.GetLength(0);
62
63              AsetaEsteet();
64          }
65
66          //Asettaa TileMappiin esteet
67          public void AsetaEsteet()
68          {
69              foreach (RoomObject ro in game.roomObjects)
70              {
71                  Rectangle rec = new Rectangle(ro.Area.Left, ro.Area.Top + 30,
72                      ro.Area.Width, ro.Area.Height - 30);
73
74                  rec = Laskenta.RectangleToTilemapRectangle(rec, TileSize);
75
76                  for (int i = 0; i < rec.Height; i++)
77                  {
78                      for (int j = 0; j < rec.Width; j++)
79                      {
80                          ruudukko[i + rec.Y, j + rec.X] = 1;
81                      }
82                  }
83              }
84          }
85
86          //Palauttaa tilen arvon
87          //0 = vapaa, 1 = este
88          public int AnnaTileArvo(int tileX, int tileY)
89          {
90              if (tileX < 0 || tileX > Leveys - 1 || tileY < 0 || tileY > Korkeus - 1)
91                  return 0;
92
93              return ruudukko[tileY, tileX];
94          }
```

```
95
96          public void AsetaEsteetJaMonsterit()...
117
118         public void AsetaEsteetJaAnsat()...
138
139         //Piirtää TileMapin
140         public void Draw(SpriteBatch spriteBatch)...
156     }
157 }
158
```

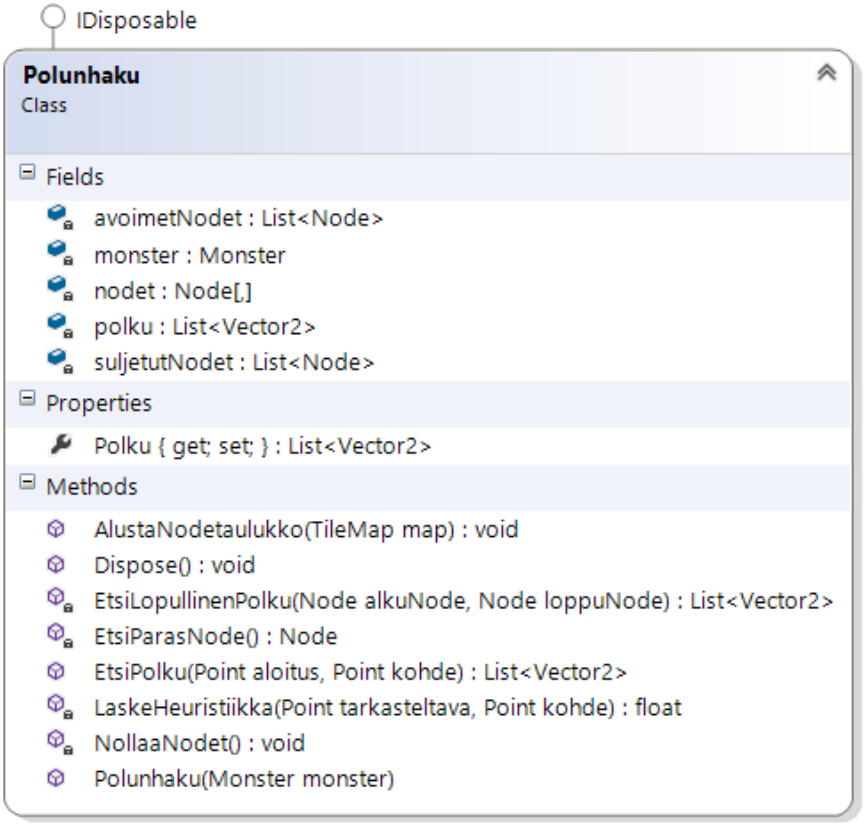
5.1.2 Node-luokka




```
7   namespace CCMonsters.CCMonster.Tekoaly.Polunhaku
8   {
9       //NODE (SOLMU)
10      //A*-polunhakualgoritmiin kuuluva luokka
11      //Muuttujat sisältävät TileMap-ruutuun asetetun solmun (node) tiedot
12
13      public class Node
14      {
15          MUUTTUJAT
43
44          OMINAISUUDET
95
96          public Node()
97          {
98              YmparoivatNodet = new List<Node>();
99          }
```

```
100
101         //Asettaa nodelle kaikki sitä ympäröivät nodet
102         public void AsetaYmparoivatNodet(Node[,] nodet, int x, int y,
103             int leveys, int korkeus, bool diagonal)
104         {
105             List<Point> nodeSijainnit = new List<Point>();
106
107             //Listataan kaikki mahdolliset ympärillä olevat nodet...
108             if (diagonal)
109             {
110                 //Käytetään 8-suuntaista tarkastelua = diagonaalinen menetelmä
111                 nodeSijainnit.Add(new Point(x, y - 1));//Ylä
112                 nodeSijainnit.Add(new Point(x, y + 1));//Ala
113                 nodeSijainnit.Add(new Point(x - 1, y));//Vasen
114                 nodeSijainnit.Add(new Point(x + 1, y));//Oikea
115                 nodeSijainnit.Add(new Point(x - 1, y - 1));//Ylävasen
116                 nodeSijainnit.Add(new Point(x - 1, y + 1));//Alavasen
117                 nodeSijainnit.Add(new Point(x + 1, y - 1));//Yläoikea
118                 nodeSijainnit.Add(new Point(x + 1, y + 1));//Alaoikea
119             }
120             else
121             {
122                 //Käytetään 4-suuntaista tarkastelua
123                 //(Vie huomattavasti vähemmän laskentatehoa kuin diagonaalinen tapa)
124                 nodeSijainnit.Add(new Point(x, y - 1));//Ylä
125                 nodeSijainnit.Add(new Point(x, y + 1));//Ala
126                 nodeSijainnit.Add(new Point(x - 1, y));//Vasen
127                 nodeSijainnit.Add(new Point(x + 1, y));//Oikea
128             }
129
130             //Käydään läpi kaikki mahdolliset ympärillä olevat nodet
131             for (int i = 0; i < nodeSijainnit.Count; i++)
132             {
133                 Point sijainti = nodeSijainnit[i];
134
135                 //Varmistetaan, että vieressä oleva node on pelikentän sisällä
136                 if (sijainti.X > 0 && sijainti.X < leveys &&
137                     sijainti.Y > 0 && sijainti.Y < korkeus)
138                 {
139                     Node viereinen = nodet[sijainti.X, sijainti.Y];
140
141                     //Jos vieressä oleva node on vapaa
142                     if (viereinen != null && viereinen.Vapaa == true)
143                     {
144                         //Lisätään viereinen node YmparoivatNodet-taulukkoon
145                         YmparoivatNodet.Add(viereinen);
146                     }
147                 }
148             }
149         }
150     }
151 }
152
```

5.2 Polunhaku-luokka (A*-algoritmi)




```
7  namespace CCMonsters.CCMonster.Tekoaly.Polunhaku
8  {
9      //A*-POLUNHAKUALGORITMIN pääluokka
10     //Muita polunhakualgoritmiin liittyviä luokkia ovat: Node, Tile, TileMap
11     //(sekä PolunHallinta, joka vastaa siitä, että monsteri kulkee laskettua polkua)
12     //
13     //Polunhakualgoritmin toiminta:
14     //
15     //1) Lisätään aloitus node  avointen nodejen listaan
16     //2) Toistetaan silmukassa:
17     //  a) Etsitään pienimmän F-arvon omaava node avointen nodejen listasta
18     //     ja asetetaan se tarkasteltavaksi nodeksi
19     //  b) Käydään läpi kaikki tarkasteltavan noden ympärillä olevat nodet:
20     //       - Hylätään, jos suljettujen listassa tai ei vapaana,
21     //         muuten seuraavat toimenpiteet:
22     //       - Jos ei avointen listassa, lisätään avointen listaan.
23     //         Asetetaan tarkastelussa oleva node isännäksi. Lasketaan F-,G- ja H-arvot
24     //       - Jos valmiiksi avointen listassa,
25     //         verrataan sen G-arvoa tarkastelussa olevan noden arvoon.
26     //         Jos arvo pienempi, matka aloituspisteestä on lyhyempi --> edullisempi reitti
27     //  c) Kun ympäröivät nodet on hoidettu,
28     //     poistetaan tarkastelussa ollut node avointen listasta
29     //     ja asetetaan se suljetuksi
30     //  d) Lopeta silmukan toistaminen kun:
31     //       -Tarkasteltava node = päätösnode (Kodetta vastaava node)
32     //3) Selvitetään käänteisessä for silmukassa lopullinen polku käyttäen isäntäsolmuja
33
34     public class Polunhaku : IDisposable
35     {
36         private Monster monster;
```

```
37
38          //Taulukko kaikista vapaista nodeista, jonka kautta hahmo voi kulkea
39          private Node[,] nodet;
40          //Lista nodeista, jotka tarkastetaan
41          private List<Node> avoimetNodet = new List<Node>();
42          //Lista nodeista, jotka on jo tarkastettu
43          private List<Node> suljetutNodet = new List<Node>();
44
45          //Lopullinen polku
46          private List<Vector2> polku;
47          public List<Vector2> Polku
48          {
49              get { return polku; }
50              set { polku = value; }
51          }
52
53          public Polunhaku(Monster monster)
54          {
55              this.polku = new List<Vector2>();
56              this.monster = monster;
57
58              AlustaNodetaulukko(monster.TileMap);
59          }
60
61          //HEURISTIIKKA
62          // Palauttaa arvion kahden pisteen välisestä etäisyydestä (H-arvo)
63          private float LaskeHeuristiikka(Point tarkasteltava, Point kohde)
64          {
65              //Käytetään arvion laskemiseen kahden pisteen välistä etäisyyttä -->
66              //(suorin/lyhyin tie "linnuntietä")
67              return (float)Math.Sqrt(Math.Pow((tarkasteltava.X - kohde.X), 2) +
68                     Math.Pow((tarkasteltava.Y - kohde.Y), 2));
69          }
70
71          //Muuttaa TileMapin node-taulukoksi ("noderuudukko"). Tämä alustus
72          //vaaditaan ennen polunhaun aloittamista
73          public void AlustaNodetaulukko(TileMap map)
74          {
75              nodet = new Node[map.Leveys, map.Korkeus];
76
77              //Jokaiselle TileMapin tilelle tehdään oma node
78              for (int i = 0; i < map.Leveys; i++)
79              {
80                  for (int j = 0; j < map.Korkeus; j++)
81                  {
82                      //Asetetaan noden sijainniksi sitä vastaavan tilen sijainti
83                      Node node = new Node();
84                      node.Sijainti = new Point(i, j);
85
86                      //Tarkastetaan tilen arvo
87                      //ja tehdään sen perusteella määritykset sitä vastaavaan nodeen
88                      //1 --> kyseisessä tilessä on este
89                      //0 --> tile on vapaa, jolloin
90                      //tallennetaan se nodejen taulukkoon tiletaulukkoa vastaavaan paikkaan
91                      if (map.AnnaTileArvo(i, j) == 0)
92                      {
93                          //koska node on vapaa, asetetaan arvoksi true
94                          node.Vapaa = true;
```

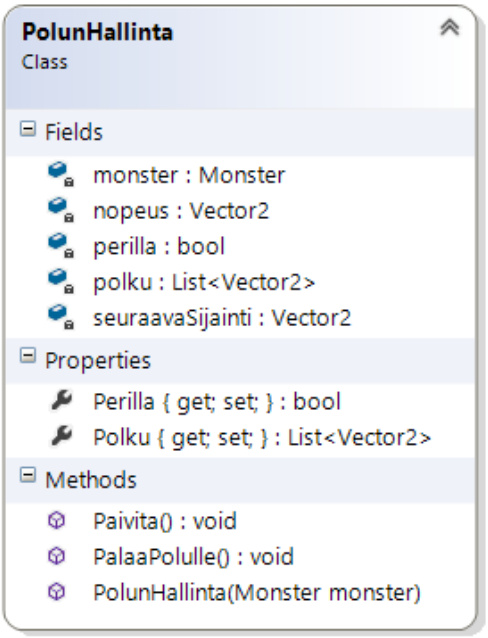
```
95
96                      //alustetaan viereisten nodejen taulukko
97                      node.YmparoivatNodet = new List<Node>();
98                      nodet[i, j] = node;
99                  }
100             }
101         }
102
103         //Jokainen nodet-taulukon node kytketään jokaiseen sen vieressä olevaan nodeen
104         for (int x = 0; x < map.Leveys; x++)
105         {
106             for (int y = 0; y < map.Korkeus; y++)
107             {
108                 Node node = nodet[x, y];
109
110                 //Tarkastellaan vain vapaana olevia nodeja
111                 //eli, jos node on esteen sitältävässä tilessä, sitä ei huomioida
112                 if (node != null && node.Vapaa == true)
113                 {
114                     node.AsetaYmparoivatNodet(nodet, x, y,
115                         map.Leveys, map.Korkeus, true);
116                 }
117             }
118         }
119     }
120
121     //Nollaa polun etsinnässä käytetyt muuttujat
122     private void NollaaNodet()
123     {
124         avoimetNodet.Clear();
125         suljetutNodet.Clear();
126
127         for (int x = 0; x < monster.TileMap.Leveys; x++)
128         {
129             for (int y = 0; y < monster.TileMap.Korkeus; y++)
130             {
131                 Node node = nodet[x, y];
132
133                 if (node != null)
134                 {
135                     node.Avoimissa = false;
136                     node.Suljetuissa = false;
137
138                     node.KuljettuMatka = float.MaxValue;
139                     node.EtaisyysKohteeseen = float.MaxValue;
140                 }
141             }
142         }
143     }
144
145     //Etsii polun alkunodesta loppunodeen käyttäen isäntänodeja
146     //Palauttaa listan vektoreista, jotka muodostavat lopullisen polun
147     private List<Vector2> EtsiLopullinenPolku(Node alkuNode, Node loppuNode)
148     {
149         suljetutNodet.Add(loppuNode);
150
151         Node isanta = loppuNode.Isanta;
```

```
152
153             while (isanta != alkuNode)
154             {
155                 suljetutNodet.Add(isanta);
156                 isanta = isanta.Isanta;
157             }
158
159             List<Vector2> finalPath = new List<Vector2>();
160
161             //Selvitetään lopullinen polkun käänteisessä for-silmukassa
162             //(lopusta alkuun käännetään alusta loppuun)
163             for (int i = suljetutNodet.Count - 1; i >= 0; i--)
164             {
165                 finalPath.Add(new Vector2(
166                     suljetutNodet[i].Sijainti.X * (int)monster.TileMap.TileSize,
167                     suljetutNodet[i].Sijainti.Y * (int)monster.TileMap.TileSize + 120));
168
169                 //+120, koska "horisontti" löytyy tästä kohdasta
170             }
171             polku = finalPath;
172             monster.AI.PathManager.Polku = finalPath;
173             return finalPath;
174         }
175
176         //Palauttaa noden, joka on lähimpänä kohdetta
177         //Kyseisellä nodella pienin F-arvo
178         private Node EtsiParasNode()
179         {
180             Node node = avoimetNodet[0];
181             float etaisyys = float.MaxValue;
182
183             //Etsitään node, joka on lähimpänä asetettua kohdetta
184             for (int i = 0; i < avoimetNodet.Count; i++)
185             {
186                 if (avoimetNodet[i].EtaisyysKohteeseen < etaisyys)
187                 {
188                     node = avoimetNodet[i];
189                     etaisyys = node.EtaisyysKohteeseen;
190                 }
191             }
192             return node;
193         }
194
195         //Etsii polun pisteestä pisteeseen.
196         //Polku = lista vektoreita,
197         //jotka muodostavat reitin aloituspisteestä kohdepisteeseen
198         public List<Vector2> EtsiPolku(Point aloitus, Point kohde)
199         {
200             // Tarkastetaan ovatko alku ja loppu samat
201             if (aloitus == kohde)
202             {
203                 //palautetaan tyhjä lista
204                 return new List<Vector2>();
205             }
206
207             //Nollataan muuttujat, siltä varalta, että niitä on jo käytetty
208             //toisen polun etsimiseen
209             NollaaNodet();
210
211             Node alkuNode = nodet[aloitus.X, aloitus.Y];
212             Node loppuNode = nodet[kohde.X, kohde.Y];
213             if (alkuNode != null && loppuNode != null)
214             {
```

```
215                 //Aseteaan arvo noden etäisyydelle kohteesta
216                 //Käytetään heuristiikkaa etäisyyden arvioinnissa
217                 alkuNode.EtaisyysKohteeseen = LaskeHeuristiikka(aloitus, kohde);
218                 //Asetetaan kuljettu matkan arvoksi 0 (G-arvo)
219                 alkuNode.KuljettuMatka = 0;
220                 //Lisätään node avointen nodejen joukkoon
221                 avoimetNodet.Add(alkuNode);
222                 alkuNode.Avoimissa = true;
223
224                 //Avointen nodejen läpikäynti...
225                 //jatketaan while-silmukkaa, kunnes avoimia nodeja ei enää löydy
226                 while (avoimetNodet.Count > 0)
227                 {
228                     //Etsitään lähimpänä kohdetta oleva node
229                     Node node = EtsiParasNode();
230
231                     //Jos nodea ei löydy, lopetetaan silmukka
232                     if (node == null)
233                     {
234                         break;
235                     }
236
237                     //Jos tarkasteltava node on sama kuin kohteen node,
238                     //selvitetään lopullinen polku
239                     if (node == loppuNode)
240                     {
241                         return EtsiLopullinenPolku(alkuNode, loppuNode);
242                     }
243
244                     // Käydään läpi jokainen tarkasteltavan noden vireinen node
245                     for (int i = 0; i < node.YmparoivatNodet.Count; i++)
246                     {
247                         Node viereinen = node.YmparoivatNodet[i];
248
249                         // Varmistetaan, että tarkastelussa oleva
250                         //vireinen node on asetettu ja että se on vapaa
251                         if (viereinen != null && viereinen.Vapaa == true)
252                         {
253                             // Lasketaan vireisen noden G-ARVO
254                             float kuljettuMatka = node.KuljettuMatka + 1;
255
256                             //Käytetään heuristiikkaa arvioidessa etäisyyttä
257                             //nodesta lopulliseen kohteeseen
258                             float heuristiikka =
259                                 LaskeHeuristiikka(viereinen.Sijainti, kohde);
260
261                             //Jos tarkastelussa oleva "viereinen" node ei ole aivointen
262                             //eikä suljettujen listassa...
263                             if (viereinen.Avoimissa == false
264                                 && viereinen.Suljetuissa == false)
265                             {
266                                 // Asetetaan laskettu G-Arvo
267                                 viereinen.KuljettuMatka = kuljettuMatka;
268                                 //Asetetaan vireisen noden F arvo käyttäen G-Arvoa
269                                 //ja heuristiikkaa
270                                 viereinen.EtaisyysKohteeseen = kuljettuMatka * heuristiikka;
271                                 //Asetetaan viereisen noden isännäksi tarkasteltava node
272                                 viereinen.Isanta = node;
273                                 //Lisätään viereinen node avointen nodejen listaan
274                                 viereinen.Avoimissa = true;
275                                 avoimetNodet.Add(viereinen);
276                             }
```

```
277                         else if (viereinen.Avoimissa || viereinen.Suljetuissa)
278                         {
279                             //Jos uusi G-arvo on vähemmän kuin viereisen noden G-arvo,
280                             //Tehdään sama kuin edellä, mutta ei lisätä nodea avoimiin
281                             if (viereinen.KuljettuMatka > kuljettuMatka)
282                             {
283                                 viereinen.KuljettuMatka = kuljettuMatka;
284                                 viereinen.EtaisyysKohteeseen =
285                                     kuljettuMatka * heuristiikka;
286 
287                                 viereinen.Isanta = node;
288                             }
289                         }
290                     }
291                 }
292                 //Poistetaan tarkasteltava node avointen listasta
293                 //ja asetetaan se suljetuksi
294                 avoimetNodet.Remove(node);
295                 node.Suljetuissa = true;
296             }
297         }
298         //Jos polkua ei löydy palautetaan tyhjä lista
299         return new List<Vector2>();
300     }
301 
302     public void Dispose()
303     {
304         GC.SuppressFinalize(this);
305     }
306 }
307 }
308 
```

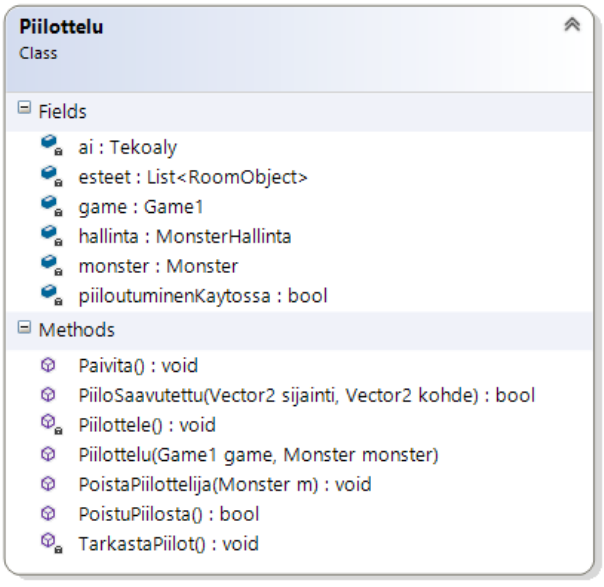
5.3 PolunHallinta-luokka



```
7  namespace CCMonsters.CCMonster.Tekoaly.Polunhaku
8  {
9      //POLUNHALLINTA
10     //Hoitaa monsterin etenemisen polulla (siirtyminen polun pisteestä toiseen)
11
12     public class PolunHallinta
13     {
14         MUUTTUJAT
22         OMINAISUUDET
39
40         public PolunHallinta(Monster monster)
41         {
42             this.monster = monster;
43             this.polku = new List<Vector2>();
44         }
45
46         public void Paivita()
47         {
48             //Jos polkua ei ole kuljettu loppuun
49             if (polku.Count > 0)
50             {
51                 //Mikäli seuraavaSijainti on nollavektori, täytyy kulkea kohti polun
52                 //ensimmäistä vektoria
53                 if (seuraavaSijainti == Vector2.Zero)
54                 {
55                     perilla = false;
56                     seuraavaSijainti = polku[0];
57                     nopeus = seuraavaSijainti - monster.Hallinta.Keskipiste;
58                     nopeus.Normalize();
59                     nopeus *= monster.Hallinta.Vauhti;
60                     monster.Hallinta.Nopeus = nopeus;
61                 }
62                 else
63                 {
64                     //Jos kuljettavaa polkua on vähintään yksi piste jäljellä
65                     if (polku.Count >= 1)
66                     {
67                         if (monster.TilanHallinta.Tila != TilanHallinta.Tilat.OttaaOsumaa &&
68                             monster.TilanHallinta.Tila != TilanHallinta.Tilat.Kuolee)
69                         {
70                             if (monster.TilanHallinta.Tila != TilanHallinta.Tilat.Piiloutuu
71                                 && monster.TilanHallinta.Tila != TilanHallinta.Tilat.
72                                 PolunHaku && monster.TilanHallinta.Tila != TilanHallinta.
73                                 Tilat.Ansassa && monster.TilanHallinta.Tila != TilanHallinta.
74                                 Tilat.KiertaaAnsaa)
75                             {
76                                 monster.TilanHallinta.AsetaTila(TilanHallinta.
77                                     Tilat.PolunHaku);
78                             }
79                             if ((seuraavaSijainti
80                                 - monster.Hallinta.Keskipiste).Length() < 10)
81                             {
82                                 polku.RemoveAt(0);
83                                 if (polku.Count > 0)
84                                 {
85                                     seuraavaSijainti = polku[0];
86                                     nopeus = seuraavaSijainti - monster.Hallinta.Keskipiste;
87                                     nopeus.Normalize();
88                                     nopeus *= monster.Hallinta.Vauhti;
89                                     monster.Hallinta.Nopeus = nopeus;
```

```
90                              }
91                              else
92                              {
93                                  nopeus = seuraavaSijainti - monster.Hallinta.Keskipiste;
94                                  nopeus.Normalize();
95                                  nopeus *= monster.Hallinta.Vauhti;
96                                  monster.Hallinta.Nopeus = nopeus;
97                              }
98                          }
99                          else
100                         {
101                             PalaaPolulle();
102                         }
103                     }
104                 }
105                 //Jos polkuun ei kuulu enään pisteitä, ollaan perillä
106                 if (polku.Count == 0 && (seuraavaSijainti
107                     - monster.Hallinta.Keskipiste).Length() < 10)
108                 {
109                     //velocity = Vector2.Zero;
110                     perilla = true;
111                     //monster.Hallinta.Nopeus = velocity;
112
113                 }
114             }
115         }
116     }
117
118     //Palauttaa monsterin takaisin polulle
119     //Antaa monsterille nopeuden, jonka suunta on kohti lähintä polun pistettä
120     public void PalaaPolulle()...
156 }
157 }
158
```

6 Piiloutuminen

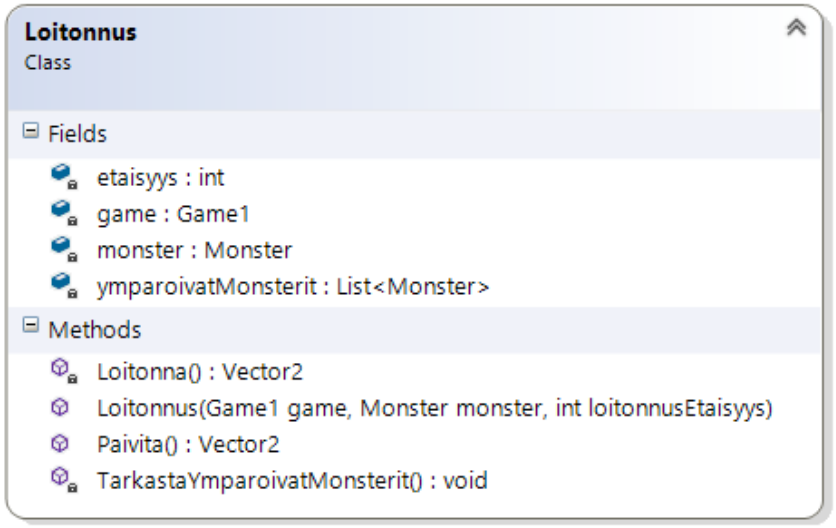


```
14  namespace CCMonsters
15  {
16      //PIILOTTELU
17      //Hoitaa monsterin piilottelu-käyttäytymismallin
18
19      public class Piilottelu
20      {
21          Game1 game;
22          List<RoomObject> esteet;
23          Monster monster;
24          MonsterHallinta hallinta;
25          Tekoaly ai;
26          bool piiloutuminenKaytossa;
27
28          MUUTTUJAT SPECIALPIILOA VARTEN
37
38          public Piilottelu(Game1 game, Monster monster)
39          {
40              this.game = game;
41              this.monster = monster;
42              this.esteet = game.roomObjects;
43              this.hallinta = monster.Hallinta;
44              this.ai = monster.AI;
45              TarkastaPiilot();
46          }
47
48          public void Paivita()
49          {
50              TarkastaPiilot();
51              Piilottele();
52          }
53
54          //Piilottelun toteuttaminen
55          private void Piilottele()
56          {
57              foreach (RoomObject ro in esteet)
58              {
59                  if (ro.Piilottelijat.Contains(monster))
60                  {
61                      //Jos piiloutuminen ei ole vielä käytössä...
62                      if (!piiloutuminenKaytossa)
63                      {
64                          //Asetetaan tilaksi Piiloutuu
65                          monster.TilanHallinta.AsetaTila(TilanHallinta.Tilat.Piiloutuu);
66                          //Etsitään polunhakualgoritmilla polku piiloon
67                          monster.AI.Polunhaku.EtsiPolku(
68                              Laskenta.VectorToTilemapPoint(hallinta.Keskipiste,
69                              monster.TileMap.TileSize),
70                              Laskenta.VectorToTilemapPoint(ro.YlareunanPiste,
71                              monster.TileMap.TileSize));
72                          //Asetetaan piiloutuminen käyttöön
73                          piiloutuminenKaytossa = true;
74                      }
75
76                      //PISTEET KOHTEESEEN KULKUA VARTEN
77                      //Keskusta (piilon takana)
78                      Vector2 pointOne = new Vector2(ro.Area.Center.X, ro.Area.Top);
79                      //Oikea puoli
80                      Vector2 pointTwo = new Vector2(ro.Area.Right +
81                          (int)monster.TileMap.TileSize, ro.Area.Top);
```

```
82                  //Vasen puoli
83                  Vector2 pointTre = new Vector2(ro.Area.Left -
84                      (int)monster.TileMap.TileSize, ro.Area.Top);
85
86                  //Satunnaismuuttuja a, joka saa arvon 2 tai 3
87                  //Käytetään liikuttaessa hahmoa satunnaisesti jommalle kummalle
88                  //puolelle piiloa
89                  Random r = new Random();
90                  int a = r.Next(2, 4);
91
92                  if (monster.TilanHallinta.Tila == TilanHallinta.Tilat.Piilossa)
93                  {
94                      ai.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.polunHaku);
95                      if (ai.KohteeseenKulku == null)
96                      {
97                          ai.LisaaKohteeseenKulku();
98                      }
99                      else
100                     {
101                         if (ai.KohteeseenKulku.Kohde != pointOne
102                             && ai.KohteeseenKulku.Kohde != pointTwo
103                             && ai.KohteeseenKulku.Kohde != pointTre)
104                         {
105                             //Siirrytään piilon taakse
106                             ai.KohteeseenKulku.Kohde = pointOne;
107                         }
108
109                         if (PiiloSaavutettu(hallinta.Keskipiste,
110                             ai.KohteeseenKulku.Kohde)
111                             && ai.KohteeseenKulku.Kohde == pointOne)
112                         {
113                             if (a == 2)
114                             {
115                                 //Siirrytään piilon oikealle puolelle
116                                 if (monster.Hallinta.Leveys / 2 < 800 - ro.Area.Right)
117                                 {
118                                     ai.KohteeseenKulku.Kohde = pointTwo;
119                                 }
120                             }
121                             else if (a == 3)
122                             {
123                                 //Siirrytään piilon vasemmalle puolelle
124                                 if (ro.Area.Left > monster.Hallinta.Leveys / 2)
125                                 {
126                                     ai.KohteeseenKulku.Kohde = pointTre;
127                                 }
128                             }
129                         }
130                         else if (PiiloSaavutettu(hallinta.Keskipiste,
131                             ai.KohteeseenKulku.Kohde)
132                             && ai.KohteeseenKulku.Kohde == pointTwo)
133                         {
134                             //Jos monsteri piilon oikealla puolella,
135                             //suoritetaan ammunta ja palataan takaisin piilon taakse
136                             monster.AI.Ammunta.Ammu();
137                             ai.KohteeseenKulku.Kohde = pointOne;
138                         }
139                         else if (PiiloSaavutettu(hallinta.Keskipiste,
140                             ai.KohteeseenKulku.Kohde)
141                             && ai.KohteeseenKulku.Kohde == pointTre)
142                         {
143                             //Jos monsteri piilon vasemmalla puolella,
144                             //suoritetaan ammunta ja palataan takaisin piilon taakse
145                             monster.AI.Ammunta.Ammu();
146                             ai.KohteeseenKulku.Kohde = pointOne;
147                         }
148                     }
149                 }
150             }
151         }
152     }
```

```
153
154         //Tarkastaa onko avoimia piiloja
155         //Jos on, lisää monsterin piilottelijaksi
156         private void TarkastaPiilot()...
171
172         //Palauttaa true, kun monsteri on saavuttanut piilon
173         //Vertaa monsterin sijaintia piilon sijaintiin - jos näiden välinen etäisyys
174         //riittävän pieni, on piilo saavutettu
175         public bool PiiloSaavutettu(Vector2 sijainti, Vector2 kohde)...
189
190         //Poistaa monsterin piilon piilottelijoista
191         public void PoistaPiilottelija(Monster m)...
202
203         //Palauttaa true, kun monster on valmis poistumaan piilosta
204         public bool PoistuPiilosta()...
227
228         //Tätä pitäisi käyttää, jos halutaan piilottelu kahden lähekkäisen objektin takana
229         //siten, että niiden molempien tarjoamat piilot katsotaan yhdeksi piiloksi
230         SpecialPiilon käyttö
303     }
304 }
305
```

7 Ympäröivien hahmojen tunnistus ja loitonnus

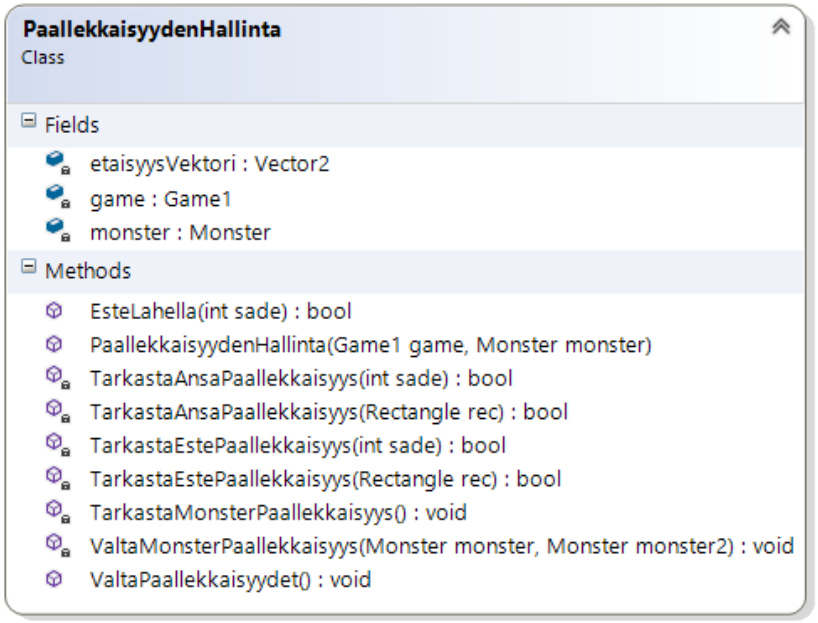



```
7   namespace CCMonsters.CCMonster.Tekoaly
8   {
9       //LOITONNUS
10      //Selvittää monsterin ympärillä tietyllä etäisyydellä olevat toiset monsterit.
11      //Mikäli monsterin läheisyydessä on toisia monstereita ohjaa monsteria kauemams näistä.
12      public class Loitonnus
13      {
14          Game1 game;
15          Monster monster;
16          List<Monster> ymparoivatMonsterit;
17          int etaisyys;
```

```
18
19          public Loitonnus(Game1 game, Monster monster, int loitonnusEtaisyys)
20          {
21              this.game = game;
22              this.monster = monster;
23              this.ymparoivatMonsterit = new List<Monster>();
24              this.etaisyys = loitonnusEtaisyys;
25          }
26
27          //Palauttaa mahdollisen vektorin,
28          //jonka mukaan monsteria on loitonnettava muista monstereista
29          public Vector2 Paivita()
30          {
31              if (monster.TilanHallinta.Tila != TilanHallinta.Tilat.Ansassa
32                  && monster.TilanHallinta.Tila != TilanHallinta.Tilat.Piilossa)
33              {
34                  TarkastaYmparoivatMonsterit();
35                  return Loitonna();
36              }
37              return Vector2.Zero;
38          }
39
40          //Selvittää kaikki lähellä olevat monsterit, joita täytyy ohjata kauemmas
41          private void TarkastaYmparoivatMonsterit()
42          {
43              ymparoivatMonsterit.Clear();
44
45              for (int i = 0; i < game.monsters.Count(); i++)
46              {
47                  if (game.monsters[i] != monster)
48                  {
49                      Vector2 etaisyysVektori =
50                          monster.Hallinta.Keskipiste - game.monsters[i].Hallinta.Keskipiste;
51
52                      //Jos etäisyys monstereiden välillä pienempi kuin määritetty etäisyys,
53                      //lisätään tarkastelussa oleva monsteri ympäröivien listaan
54                      if (etaisyysVektori.Length() < etaisyys)
55                      {
56                          ymparoivatMonsterit.Add(game.monsters[i]);
57                      }
58                  }
59              }
60          }
61
62          //Laskee vektorin,
63          //jonka mukaan monsteria on loitonnettava sitä ympäröivistä monstereista
64          private Vector2 Loitonna()
65          {
66              //vektori, jonka mukaan monsteria täytyy loitontaa ympäröivistä monstereista
67              Vector2 ohjausVektori = Vector2.Zero;
68              //Käydään läpi kaikki ympärillä olevat monsterit
69              for (int i = 0; i < ymparoivatMonsterit.Count; i++)
70              {
71                  //Selvitetään vektori monstereiden välillä
72                  Vector2 etaisyysVektori = monster.Hallinta.Keskipiste -
73                      ymparoivatMonsterit[i].Hallinta.Keskipiste;
74
```

```
75                  //ohjausvektori on kääntäen verrannollinen monstereiden väliseen
76                  //etäisyyteen nähden
77                  ohjausVektori += Vector2.Normalize(etaisyysVektori)
78                      / etaisyysVektori.Length();
79              }
80              //palautetaan lopullinen ohjausvoima,
81              //jossa on otettu huomioon kaikki ympäröivät monsterit
82              return ohjausVektori;
83          }
84      }
85  }
86
```

8 Päällekkäisyyden hallinta

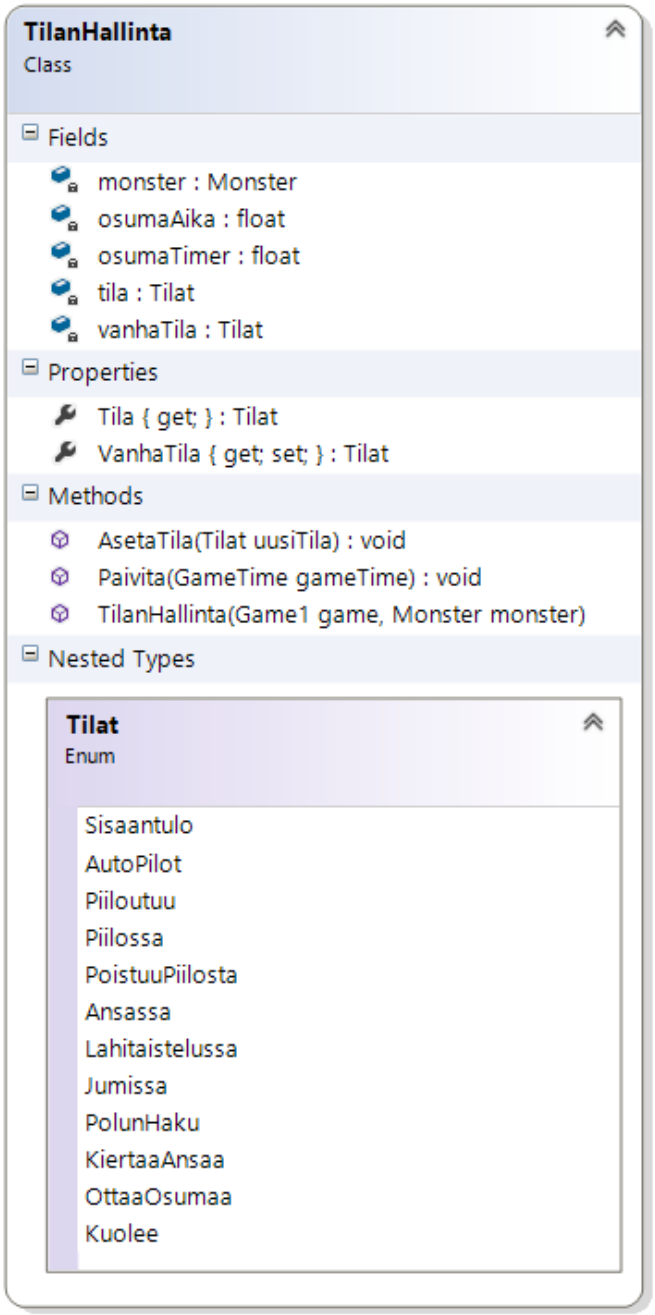



```
13  namespace CCMonsters.CCMonster.Tekoaly
14  {
15      //PÄÄLLEKKÄISYYDEN HALLINTA
16      //Luokan metodien tehtävänä on tarkastaa mahdolliset päällekkäisyydet esteiden, asnsojen
17      //ja muiden monstereiden kanssa sekä tehdä tarvittavat toimenpiteet päällekkäisyyksien
18      //välttämiseksi.
19
20      public class PaallekkaisyydenHallinta
21      {
22          private Monster monster;
23          private Game1 game;
24          private Vector2 etaisyysVektori;
25
26          public PaallekkaisyydenHallinta(Game1 game, Monster monster)
27          {
28              this.monster = monster;
29              this.game = game;
30              this.etaisyysVektori = Vector2.Zero;
31          }
```

```
32
33          //Pelilogiikan päivityksen yhteydessä tarkastetaan kaikki
34          //mahdolliset päällekkäisyydet. Mikäli päällekkäisyyttä esiintyy, muutetaan
35          //monsterin sijaintia.
36          public void ValtaPaallekkaisyydet()
37          {
38              if (TarkastaEstePaallekkaisyys(10) && etaisyysVektori != Vector2.Zero)
39              {
40                  monster.Hallinta.Sijainti = (monster.Hallinta.Sijainti
41                      + (Vector2.Normalize(etaisyysVektori)) * 3f);
42              }
43              if (TarkastaAnsaPaallekkaisyys(1) && etaisyysVektori != Vector2.Zero)
44              {
45                  monster.Hallinta.Sijainti = (monster.Hallinta.Sijainti
46                      + (Vector2.Normalize(etaisyysVektori)) * 1f);
47              }
48
49              TarkastaMonsterPaallekkaisyys();
50          }
51
52          //Selvittää onko tarksteltava monsteri päällekkäin jonkin esteen kanssa
53          ESTEEN JA MONSTERIN PÄÄLLEKKÄISYYS
121
122         //Selvittää onko tarksteltava monsteri päällekkäin jonkin ansan kanssa
123         ANSAN JA MONSTERIN PÄÄLLEKKÄISYYS
190
191         //Selvittää onko tarksteltava monsteri päällekkäin jonkin toisen monsterin kanssa
192         #region MONSTERIN JA MONSTERIN PÄÄLLEKKÄISYYS
193
194         private void TarkastaMonsterPaallekkaisyys()
195         {
196             for (int i = 0; i < game.monsters.Count(); i++)
197             {
198                 if (game.monsters[i] != monster)
199                 {
200                     if (game.monsters[i].TilanHallinta.Tila != TilanHallinta.Tilat.Ansassa)
201                     {
202                         if (monster.MonsterTyyppi == Monster.MonsterTyypit.robo)
203                         {
204                             //Jos monsterin tyyppi on robo,
205                             //vain, jos ollaan väistettävän takana
206                             if (monster.alareunaY < game.monsters[i].alareunaY)
207                             {
208                                 ValtaMonsterPaallekkaisyys(monster, game.monsters[i]);
209                             }
210                         }
211                         else
212                         {
213                             if (game.monsters[i].MonsterTyyppi != Monster.MonsterTyypit.robo)
214                             {
215                                 ValtaMonsterPaallekkaisyys(monster, game.monsters[i]);
216                             }
217                         }
218                     }
219                 }
220             }
221         }
```

```
222         private void ValtaMonsterPaallekkaisyys(Monster monster, Monster monster2)
223         {
224             //vektori vihollisten välillä
225             Vector2 etaisyysVektori =
226                 monster.Hallinta.Keskipiste - monster2.Hallinta.Keskipiste;
227
228             float etaisyys = etaisyysVektori.Length();
229
230             //Jos etäisyys monsterin ja
231             //esteen välillä on pienempi kuin niiden säteiden summa,
232             //monsteria täytyy siirtää etäisyysVektorin kanssa yhdensuuntaisesti
233             float paallekkaisyys = monster2.Hallinta.RajaSade +
234                 monster.Hallinta.RajaSade - etaisyys;
235
236             if (paallekkaisyys >= 0)
237             {
238                 //siirretään vihollista kauemmas toisesta päälekkäisyyden verran
239                 //pitää tarkemmin määrittää mihin suuntaan siirretään?
240                 monster.Hallinta.Sijainti = (monster.Hallinta.Sijainti +
241                     (etaisyysVektori / etaisyys) * paallekkaisyys);
242             }
243         }
244         #endregion
245     }
246 }
247
```

9 Päätöksenteko ja tilanhallinta




```
7  namespace CCMonsters
8  {
9      //TILAKONE
10     //Luokan tehtävänä on hallita monsterin tilaa
11     //Monsterin tila määrää mm. mitä animaatiota käytetään ja
12     //mitä käyttäytymismallia (tekoäly-luokka) suorittetaan
13
14     public class TilanHallinta
15     {
16         Monster monster;
```

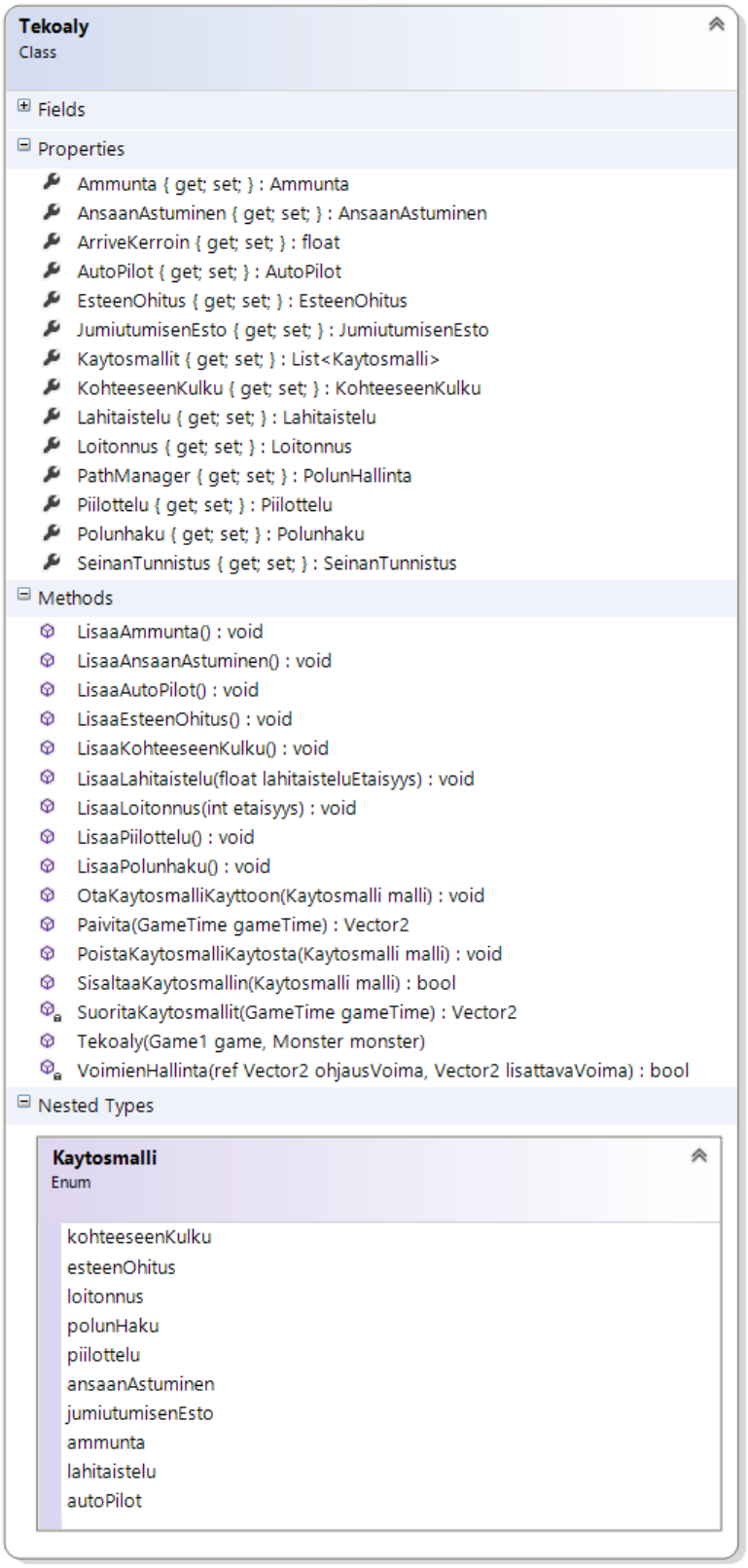
```
17
18          //Siirryttäessä uuteen tilaan, nykyinen tila tallentuu vanhaTila-muuttujaan
19          //Käytetään palatessa takaisin edelliseen tilaan
20          private Tilat vanhaTila;
21
22          //Muuttujat OttaaOsumaa-tilaa varten:
23          private float osumaTimer;//aika, kuinka kauan on ottanut osumaa
24          private float osumaAika;//aika, kuinka kauan ottaa osumaa
25
26          public Tilat VanhaTila
27          {
28              get { return vanhaTila; }
29              set
30              {
31                  vanhaTila = value;
32              }
33          }
34
35          //Tilakoneen Tilat
36          public enum Tilat
37          {
38              Sisaantulo,
39              AutoPilot,
40              Piiloutuu,
41              Piilossa,
42              PoistuuPiilosta,
43              Ansassa,
44              Lahitaistelussa,
45              Jumissa,
46              PolunHaku,
47              KiertaaAnsaa,
48              OttaaOsumaa,
49              Kuolee
50          }
51
52          //Asetetaan oletustulaksi sisääntulo, koska se on
53          //jokaisen monsterin ensimmäinen tila
54          private Tilat tila = Tilat.Sisaantulo;
55          public Tilat Tila
56          {
57              get
58              {
59                  return tila;
60              }
61          }
62          public TilanHallinta(Game1 game, Monster monster)
63          {
64              this.monster = monster;
65              this.osumaTimer = 0;
66              this.osumaAika = 200;
67          }
68
69          //AsetaTila-metodi
70          //Uuden tilan asettaminen tapahtuu aina tämän metodin kautta
71          //Saa parametrina tilan, johon siirrytään
72          public void AsetaTila(Tilat uusiTila)
73          {
74              if (tila != Tilat.Kuolee)
75              {
```

```
76                  switch (uusiTila)
77                  {
78                      case (Tilat.Sisaantulo):
79                          tila = Tilat.Sisaantulo;
80                          break;
81                      case (Tilat.AutoPilot):
82                          VanhaTila = tila;
83                          tila = Tilat.AutoPilot;
84                          break;
85                      case (Tilat.OttaaOsumaa):
86                          if (tila != Tilat.OttaaOsumaa)
87                          {
88                              if (tila != Tilat.Ansassa)
89                              {
90                                  VanhaTila = tila;
91                              }
92                          }
93                          if (tila != Tilat.Ansassa)
94                          {
95                              tila = Tilat.OttaaOsumaa;
96                          }
97                          break;
98                      case (Tilat.Kuolee):
99                          //VanhaTila = tila;
100                         tila = Tilat.Kuolee;
101                         break;
102                     case (Tilat.Piilossa):
103                         VanhaTila = tila;
104                         tila = Tilat.Piilossa;
105                         break;
106                     case (Tilat.Piiloutuu):
107                         VanhaTila = tila;
108                         tila = Tilat.Piiloutuu;
109                         monster.AI.OtaKaytosmalliKayttoon(Tekoaly.Kaytosmalli.polunHaku);
110                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.autoPilot);
111                         break;
112                     case (Tilat.PoistuuPiilosta):
113                         VanhaTila = tila;
114                         tila = Tilat.PoistuuPiilosta;
115                         break;
116                     case (Tilat.Ansassa):
117                         VanhaTila = tila;
118                         tila = Tilat.Ansassa;
119                         monster.AI.PoistaKaytosmalliKaytosta(CCMonsters.Tekoaly.Kaytosmalli.
120                             polunHaku);
121                         monster.AI.PoistaKaytosmalliKaytosta(CCMonsters.Tekoaly.Kaytosmalli.
122                             autoPilot);
123                         break;
124                     case (Tilat.Jumissa):
125                         VanhaTila = tila;
126                         tila = Tilat.Jumissa;
127                         break;
128                     case (Tilat.KiertaaAnsaa):
129                         VanhaTila = tila;
130                         tila = Tilat.KiertaaAnsaa;
131                         break;
132                     case (Tilat.PolunHaku):
133                         VanhaTila = tila;
134                         tila = Tilat.PolunHaku;
```

```
135                     if (monster.AI.SisaltaaKaytosmallin(Tekoaly.Kaytosmalli.
136                         kohteeseenKulku))
137                     {
138                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.
139                             kohteeseenKulku);
140                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.
141                             autoPilot);
142                     }
143                     break;
144                 case (Tilat.Lahitaistelussa):
145                     VanhaTila = tila;
146                     tila = Tilat.Lahitaistelussa;
147                     break;
148             }
149         }
150     }
151 
152     //Paivita-metodi hoitaa tilan päivityksen/tarkastuksen
153     //kutsutaan monster-olion update-metodikutsun yhteydessä
154     public void Paivita(GameTime gameTime)
155     {
156         switch (tila)
157         {
158             case (Tilat.AutoPilot):
159                 monster.AI.OtaKaytosmalliKayttoon(Tekoaly.Kaytosmalli.autoPilot);
160                 break;
161             case (Tilat.OttaaOsumaa):
162                 if (monster.Health <= 0)
163                 {
164                     AsetaTila(Tilat.Kuolee);
165                 }
166                 else
167                 {
168                     osumaTimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
169                     if (osumaTimer > osumaAika)
170                     {
171                         osumaTimer = 0;
172                         AsetaTila(vanhaTila);
173                     }
174                 }
175                 break;
176             case (Tilat.Piilossa):
177                 monster.AI.Ammunta.Ampuu = false;
178                 if (monster.BulletAmount <= monster.BulletUsed)
179                 {
180                     AsetaTila(Tilat.PoistuuPiilosta);
181                 }
182                 break;
183             case (Tilat.Piiloutuu):
184                 if (monster.AI.PathManager.Perilla)
185                 {
186                     AsetaTila(Tilat.Piilossa);
187                     if (!monster.AI.SisaltaaKaytosmallin(Tekoaly.Kaytosmalli.
188                         kohteeseenKulku))
189                     {
190                         monster.AI.OtaKaytosmalliKayttoon(Tekoaly.Kaytosmalli.
191                             kohteeseenKulku);
192                     }
193                 }
194                 break;
```

```
195                 case (Tilat.PoistuuPiilosta):
196                     if (monster.AI.Piilottelu.PoistuPiilosta())
197                     {
198                         monster.AI.Piilottelu.PoistaPiilottelija(monster);
199                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.piilottelu);
200                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.
201                             kohteeseenKulku);
202                         monster.AI.Polunhaku.EtsiPolku(Laskenta.VectorToTilemapPoint(
203                             monster.Hallinta.Keskipiste, monster.TileMap.TileSize),
204                             Laskenta.VectorToTilemapPoint(new Vector2(400, 460),
205                             monster.TileMap.TileSize));
206                         this.tila = Tilat.PolunHaku;
207                     }
208                     break;
209                 case (Tilat.Jumissa):
210                     //monster.AI.OtaKaytosmalliKayttoon(Tekoaly.Kaytosmalli.polunHaku);
211                     //monster.TileMap.AsetaEsteetJaMonsterit();
212                     //monster.AI.Polunhaku.Initialize(monster.TileMap);
213                     //monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.monsterVaisto)
214                     //monster.AI.Polunhaku.EtsiPolku(Laskenta.VectorToTilemapPoint(
215                     //monster.Hallinta.Keskipiste, monster.TileMap.TileSize),
216                     //Laskenta.VectorToTilemapPoint(new Vector2(400, 460),
217                     //monster.TileMap.TileSize));
218                     //this.tila = Tilat.Polunhaku;
219                     break;
220                 case (Tilat.PolunHaku):
221                     if (monster.AI.PathManager.Perilla)
222                     {
223                         monster.AI.PoistaKaytosmalliKaytosta(Tekoaly.Kaytosmalli.polunHaku);
224                         monster.AI.OtaKaytosmalliKayttoon(Tekoaly.Kaytosmalli.
225                             kohteeseenKulku);
226                         monster.AI.KohteeseenKulku.Kohde = new Vector2(400, 460);
227                     }
228                     break;
229                 case (Tilat.Ansassa):
230 
231                     break;
232                 case (Tilat.KiertaaAnsaa):
233                     break;
234             }
235         }
236     }
237 }
238 
```

10 Tekoaly-luokka



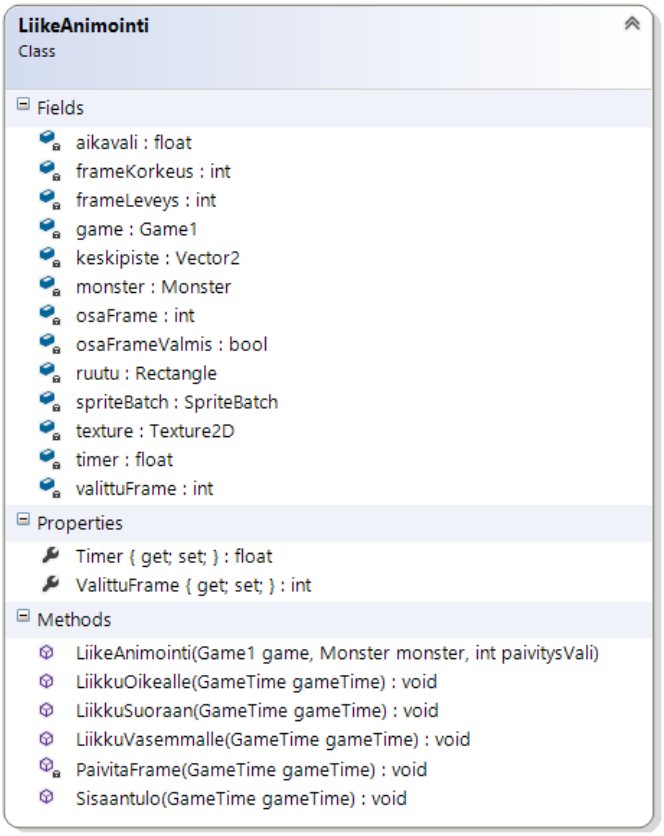
```
8   namespace CCMonsters
9   {
10      //TEKOÄLY
11      //Luokka hoitaa teoälyyn liittyvän hallinnoinnin
12      //Sisältää monsterin käyttäytymismallit ja niiden päivityksen
13
14      public class Tekoaly
15      {
16          MUUTTUJAT
56
57          OMINAISUUDET (getterit+setterit)
131
132         public Tekoaly(Game1 game, Monster monster)
133         {
134             this.game = game;
135             this.monster = monster;
136             this.kaytosmallit = new List<Kaytosmalli>();
137             this.seinanTunnistus = new SeinanTunnistus(game, monster);
138         }
139         public bool SisaltaaKaytosmallin(Kaytosmalli malli)
140         {
141             if (this.kaytosmallit.Contains(malli))
142             {
143                 return true;
144             }
145             return false;
146         }
147
148         public Vector2 Paivita(GameTime gameTime)
149         {
150             if (monster.TilanHallinta.Tila != TilanHallinta.Tilat.Sisaantulo)
151             {
152                 return SuoritaKaytosmallit(gameTime);
153             }
154             return Vector2.Zero;
155         }
156
157         //Suorittaa kaikki käytössä olevat käyttäytymismallit
158         private Vector2 SuoritaKaytosmallit(GameTime gameTime)
159         {
160             //Vektori, joka on kaikkien ohjaukseen vaikuttavien käyttäytymismallien summa
161             Vector2 ohjausVoima = Vector2.Zero;
162             Vector2 force = Vector2.Zero;
163
164             //Seiniä täytyy väistää aina
165             seinanTunnistus.ValtaSeina();
166
167             //KOHTEESEEN KULKU
168             if (this.SisaltaaKaytosmallin(Kaytosmalli.kohteeseenKulku))
169             {
170                 force += KohteeseenKulku.LaskeOhjaus(monster.Hallinta) * arriveKerroin;
171                 if (!VoimienHallinta(ref ohjausVoima, force))
172                     return ohjausVoima;
173             }
```

```
174
175             //ESTEEN OHITUS ympyrän kaarta hyödyntäen
176             if (this.SisaltaaKaytosmallin(Kaytosmalli.esteenOhitus))
177             {
178                 force += EsteenOhitus.LaskeOhjaus(monster.Hallinta, this)
179                     * esteenOhitusKerroin;
180                 if (!VoimienHallinta(ref ohjausVoima, force))
181                     return ohjausVoima;
182             }
183
184             //LOITONNUS
185             if (this.SisaltaaKaytosmallin(Kaytosmalli.loitonnus))
186             {
187                 ohjausVoima += loitonnus.Paivita();
188                 //if (!VoimienHallinta(ref ohjausVoima, force, monster.Hallinta))
189                 //    return ohjausVoima;
190             }
191
192             //PIILOTTELU esteen takana
193             if (this.SisaltaaKaytosmallin(Kaytosmalli.piilottelu))
194             {
195                 Piilottelu.Paivita();
196             }
197
198             //AMMUNTA
199             if (this.SisaltaaKaytosmallin(Kaytosmalli.ammunta))
200             {
201                 Ammunta.Update(gameTime);
202             }
203
204             //A*-polunhakualgoritmin käyttö
205             //(Päivitys tapahtuu monster-luokan Update()-metodikutsun yhteydessä)
206             if (this.SisaltaaKaytosmallin(Kaytosmalli.polunHaku))
207             {
208                 //PathManager.Paivita();
209             }
210
211             //AUTOPILOT (monsterin automaattinen liikkuminen pelimaailmassa)
212             if (this.SisaltaaKaytosmallin(Kaytosmalli.autoPilot))
213             {
214                 AutoPilot.Paivita();
215             }
216
217             //ANSAN TUNNISTUS
218             if (this.SisaltaaKaytosmallin(Kaytosmalli.ansaanAstuminen))
219             {
220                 AnsaanAstuminen.TarkastaAnsaanAstuminen(30);
221                 AnsaanAstuminen.Paivita();
222             }
223
224             //LÄHITAISTELU
225             if (this.SisaltaaKaytosmallin(Kaytosmalli.lahitaistelu))
226             {
227                 lahitaistelu.Paivita();
228             }
229
230             //Jumiutumisen välttäminen
231             if (this.SisaltaaKaytosmallin(Kaytosmalli.jumiutumisenEsto))
232             {
233                 //jumiutumisenEsto.TarkastaJumiutuminen(gameTime);
234             }
```

```
235
236             return ohjausVoima;
237         }
238
239         METODIT KÄYTÖSMALLIEN LISÄÄMISEEN
289
290         public void PoistaKaytosmalliKaytosta(Kaytosmalli malli)
291         {
292             if (SisaltaaKaytosmallin(malli))
293             {
294                 kaytosmallit.Remove(malli);
295             }
296         }
297         public void OtaKaytosmalliKayttoon(Kaytosmalli malli)
298         {
299             if (!SisaltaaKaytosmallin(malli))
300             {
301                 if (malli == Kaytosmalli.kohteeseenKulku)
302                 {
303                     PoistaKaytosmalliKaytosta(Kaytosmalli.polunHaku);
304                     //PoistaKaytosmalliKaytosta(Kaytosmalli.autoPilot);
305                 }
306                 if (malli == Kaytosmalli.polunHaku)
307                 {
308                     PoistaKaytosmalliKaytosta(Kaytosmalli.kohteeseenKulku);
309                     PoistaKaytosmalliKaytosta(Kaytosmalli.autoPilot);
310                 }
311                 kaytosmallit.Add(malli);
312             }
313         }
314         private bool VoimienHallinta(ref Vector2 ohjausVoima, Vector2 lisattavaVoima)...
342     }
343 }
344
```

11 Animaatiot

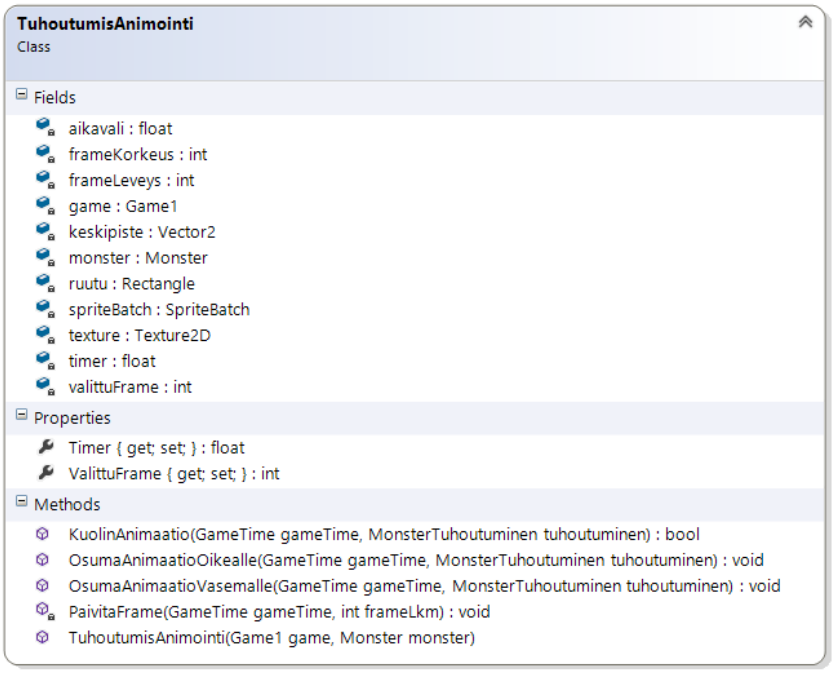
11.1 Liikkumisen animaatiot




```
12  namespace CCMonsters
13  {
14      //LIIKEANIMOINTI
15      //Vastaa monsterin liikkumisen animaatioista
16      //Animaatiot: sisääntulo, suoraan, vasemmalle, oikealle
17      public class LiikeAnimointi
18      {
19          MUUTTUJAT
38
39          OMINAISUUDET
51
52          public LiikeAnimointi(Game1 game, Monster monster, int paivitysVali)
53          {
54              this.game = game;
55              this.monster = monster;
56              this.texture = monster.MonsterTexture;
57              this.spriteBatch = new SpriteBatch(game.GraphicsDevice);
58
59              this.valittuFrame = 0; //Parhaillaan näytettävä frame spritesheetistä
60              this.aikavali = paivitysVali; //Framen päivitysväli (siirtyminen toiseen frameen)
61              this.frameLeveys = (int)monster.Hallinta.Leveys;
62              this.frameKorkeus = (int)monster.Hallinta.Korkeus;
63              this.osaFrame = 0; //Sisääntuloanimaatiossa ensimmäinen frame on jaettu osiin
64          }
```

```
65
66          //Sisääntulon animaatio
67          public void Sisaantulo(GameTime gameTime)...
111
112         //Suoraan kulkemisen animaatio
113         public void LiikkuSuoraan(GameTime gameTime)...
124
125         //Vasemmalle kulkemisen animaatio
126         public void LiikkuVasemmalle(GameTime gameTime)
127         {
128             PaivitaFrame(gameTime);
129             //näytettävä kuva (frame) spritesheetistä
130             ruutu = new Rectangle(
131                 valittuFrame * frameLeveys, frameKorkeus, frameLeveys, frameKorkeus);
132             //kuvan keskipiste
133             keskipiste = new Vector2(ruutu.Width / 2, ruutu.Height / 2);
134
135             spriteBatch.Begin();
136             //piirto
137             spriteBatch.Draw(texture, monster.Hallinta.Keskipiste, ruutu, Color.White, 0f,
138                 keskipiste, Laskenta.LaskeSpriteSkaalaus(monster.Hallinta.Keskipiste),
139                 SpriteEffects.None, 0);
140             spriteBatch.End();
141         }
142
143         //Oikealle kulkemisen animaatio
144         public void LiikkuOikealle(GameTime gameTime)...
153
154         //Siirtyminen spritesheetin kuvasta toiseen
155         private void PaivitaFrame(GameTime gameTime)
156         {
157             timer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
158             if (timer > aikavali)
159             {
160                 valittuFrame++;
161                 timer = 0;
162                 if (valittuFrame >= monster.NumberOfFramesX)
163                 {
164                     valittuFrame = 0;
165                 }
166             }
167         }
168     }
169 }
170
```

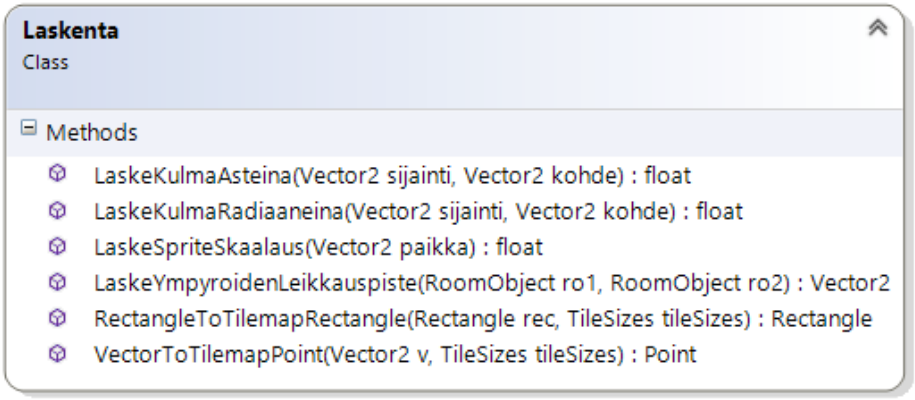
11.2 Tuhoutumisen animaatiot




```
8   namespace CCMonsters
9   {
10      public class TuhoutumisAnimointi
11      {
12          Game1 game;
13          Monster monster;
14          SpriteBatch spriteBatch;
15          Texture2D texture;
16
17          //SpriteSheetin käsittelyyn liittyvät muuttujat
18          Rectangle ruutu;//SpriteSheetin osa, joka näytetään animaation framena
19          int valittuFrame;//Parhaillaan näytettävä spritesheetin frame
20          int frameLeveys;//framen leveys
21          int frameKorkeus;//framen korkeus
22          Vector2 keskipiste;
23          float timer;//Ajastin, jonka mukaan framea vaihdetaan
24          float aikavali;//Aika, jonka jälkeen siirrytään näyttämään seuraavaa framea
25
26          public float Timer
27          {
28              get { return timer; }
29              set { timer = value; }
30          }
31          public int ValittuFrame
32          {
33              get { return valittuFrame; }
34              set { valittuFrame = value; }
35          }
36
37          public TuhoutumisAnimointi(Game1 game, Monster monster)...
49
50          //Kuoleman animaatio
51          #region PERUSKUOLEMA
```

```
52          public bool KuolinAnimaatio(GameTime gameTime, MonsterTuhoutuminen tuhoutuminen)
53          {
54              if (valittuFrame < tuhoutuminen.NumberOfFrames-1)
55              {
56                  aikavali = 40;
57                  PaivitaFrame(gameTime, tuhoutuminen.NumberOfFrames);
58                  ruutu = new Rectangle(valittuFrame * tuhoutuminen.Width, 0,
59                      tuhoutuminen.Width, tuhoutuminen.Height);
60                  keskipiste = new Vector2(ruutu.Width / 2, ruutu.Height / 2);
61                  spriteBatch.Begin();
62                  spriteBatch.Draw(tuhoutuminen.TuhoutumisTexture, monster.Hallinta.Keskipiste,
63                      ruutu, Color.White, 0f, keskipiste, Laskenta.LaskeSpriteSkaalaus(monster.
64                      Hallinta.Keskipiste), SpriteEffects.None, 0);
65                  spriteBatch.End();
66                  return false;
67              }
68              else
69              {
70                  return true;
71              }
72          }
73          #endregion
74
75          //Osumien animaatiot eri suunntiin
76          #region OSUMAN ANIMAATIOT
77          public void OsumaAnimaatioVasemalle(GameTime gameTime,
78              MonsterTuhoutuminen tuhoutuminen)
79          {
80              valittuFrame = 0;
81              ruutu = new Rectangle(valittuFrame * tuhoutuminen.Width, 0, tuhoutuminen.Width,
82                  tuhoutuminen.Height);
83              keskipiste = new Vector2(ruutu.Width / 2, ruutu.Height / 2);
84              spriteBatch.Begin();
85              spriteBatch.Draw(tuhoutuminen.TuhoutumisTexture, monster.Hallinta.Keskipiste,
86                  ruutu, Color.White, 0f, keskipiste, Laskenta.LaskeSpriteSkaalaus(monster.
87                  Hallinta.Keskipiste), SpriteEffects.None, 0);
88              spriteBatch.End();
89          }
90          public void OsumaAnimaatioOikealle(GameTime gameTime,
91              MonsterTuhoutuminen tuhoutuminen)...
100         #endregion
101
102         private void PaivitaFrame(GameTime gameTime, int frameLkm)...
116     }
117 }
118
```

12 Laskenta




```
8  namespace CCMonsters
9  {
10     //LASKENTA
11     //Tähän luokkaan on kerätty staattisia metodeja,
12     //jotka hoitavat tekoälyn usein käytetyt sekä vaativimmat laskutoimitukset
13
14     public class Laskenta
15     {
16         //Palauttaa kulman suuruudeen radiaaneina
17         public static float LaskeKulmaRadiaaneina(Vector2 sijainti, Vector2 kohde)
18         {
19             return (float)System.Math.Atan2(sijainti.Y - kohde.Y, sijainti.X - kohde.X);
20         }
21         //Palauttaa kulman suuruudeen asteina
22         public static float LaskeKulmaAsteina(Vector2 sijainti, Vector2 kohde)
23         {
24             float alfa = (float)Math.Atan2(sijainti.Y - kohde.Y, sijainti.X - kohde.X);
25             alfa = (float)(alfa * (180 / Math.PI));
26             return alfa;
27         }
28
29         //Palauttaa kahden ympyrän leikkauspisteen.
30         //Parametrina roomobjektit, joiden ympyröiden leikkauspistettä tarkastellaan.
31         public static Vector2 LaskeYmpyroidenLeikkauspiste(RoomObject ro1, RoomObject ro2)
32         {
33             Vector2 etaisyys = ro1.Keskipiste - ro2.Keskipiste;
34             float sateidenSumma = ro1.Sade + ro2.Sade;
35
36             //ESTEIDEN YHTEINEN KESKIPISTE
37             if (etaisyys.Length() < sateidenSumma)
38             {
39                 double b = Math.Pow(ro2.Area.Width / 2, 2) + Math.Pow(etaisyys.Length(), 2)
40                     - Math.Pow(ro1.Area.Width / 2, 2);
41                 b = b / (2 * etaisyys.Length());
42                 double a = etaisyys.Length() - b;
43
44                 //Keskipisteen ja leikkauspisteen välinen korkeus
45                 double korkeus = Math.Sqrt((Math.Pow(ro1.Area.Width / 2, 2)
46                     - Math.Pow(a, 2)));
47
48                 //Keskipisteiden  välisen JANAN JAKOPISTE
49                 double x0 = (b * ro1.Keskipiste.X + a * ro2.Keskipiste.X) / (b + a);
50                 double y0 = (b * ro1.Keskipiste.Y + a * ro2.Keskipiste.Y) / (b + a);
51                 Vector2 jakoPiste = new Vector2((float)x0, (float)y0);
52
```

```
53                  //LEIKKAUSPISTE
54                  //x-koordinaatti
55                  double x3 = x0 - korkeus * (ro1.Keskipiste.Y - ro2.Keskipiste.Y)
56                      / etaisyys.Length();
57                  //y-koordinaatti
58                  double y3 = y0 + korkeus * (ro1.Keskipiste.X - ro2.Keskipiste.X)
59                      / etaisyys.Length();
60
61                  Vector2 leikkausPiste = new Vector2((float)x3, (float)y3);
62                  //Ympyröillä on mahdollisesti myös toinen leikkauspiste,
63                  //mutta sitä ei tarvitse huomioida, koska monsterit kulkevat ylhäältä
64                  //alaspäin ja toinen piste jää edellä lasketun alapuolelle
65
66                  return leikkausPiste;
67              }
68              return Vector2.Zero;
69          }
70
71          //TilemMap-ruudukkoa käytettäessä vektorit täytyy kääntää ruudokkoon
72          //Se mihin ruutuun vektori osuu, riippuu ruudun koosta
73          public static Point VectorToTilemapPoint(Vector2 v, Tile.TileSizes tileSizes)
74          {
75              int x = (int)(v.X / (int)tileSizes);
76              int y = (int)((v.Y - 120) / (int)tileSizes);//horisontti 120
77              if (y <= 0)
78              {
79                  y = 0;
80                  return new Point(x, y);
81              }
82              return new Point(x, y);
83          }
84
85          //TileMap-ruudukkoa käytettäessä Recktanglet täytyy kääntää ruudukkoon sopivaksi
86          //Rectanglen arvot määräytyvät ruudun (tileSize) koon mukaan
87          public static Rectangle RectangleToTilemapRectangle(Rectangle rec,
88              Tile.TileSizes tileSizes)
89          {
90              int x = (int)Math.Floor(rec.X / (int)tileSizes) - 1;
91              int y = (int)Math.Floor((rec.Y - 120) / (int)tileSizes) + 1;
92              int leveys = (int)Math.Floor(rec.Width / (int)tileSizes) + 2;
93              int korkeus = (int)Math.Floor(rec.Height / (int)tileSizes);
94
95              //Varmistetaan että koordinaatit alueen sisällä
96              if (x < 0)
97              {
98                  x = 0;
99              } if (x > 800)
100             {
101                 x = 800;
102             }
103             return new Rectangle(x, y, leveys, korkeus);
104         }
105
106         //Skaalaa spriten etäisyyden mukaan oikean kokoiseksi.
107         //Palauttaa tarvittavan skaalauksen. parametrina piirtokohdan sijainti
108         public static float LaskeSpriteSkaalaus(Vector2 paikka)
109         {
110             //Skaalaus on 1, kun monsterin paikan (keskipiste) y-koordinaatti > 460
111             //Skaalaus on 0.85, kun monsterin paikan (keskipiste) y-koordinaatti < 130
112             //Selvitetään skaalaus, kun 130 < paikka < 460
113             float skaalaus = 1;
114             int alaraja = 440;
```

```
115             int ylaraja = 130;
116             float ylarajaSkaalaus = 0.85f;
117             float alarajaSkaalaus = 1f;
118 
119             if (paikka.Y >= alaraja)
120             {
121                 skaalaus = 1;
122                 return skaalaus;
123             }
124             else if (paikka.Y <= ylaraja)
125             {
126                 return 0.85f;
127             }
128             else if (paikka.Y > ylaraja && paikka.Y < alaraja)
129             {
130                 skaalaus = ylarajaSkaalaus + ((alarajaSkaalaus - ylarajaSkaalaus)
131                     * paikka.Y / (alaraja - ylaraja));
132                 return skaalaus;
133             }
134             return skaalaus;
135         }
136     }
137 }
```